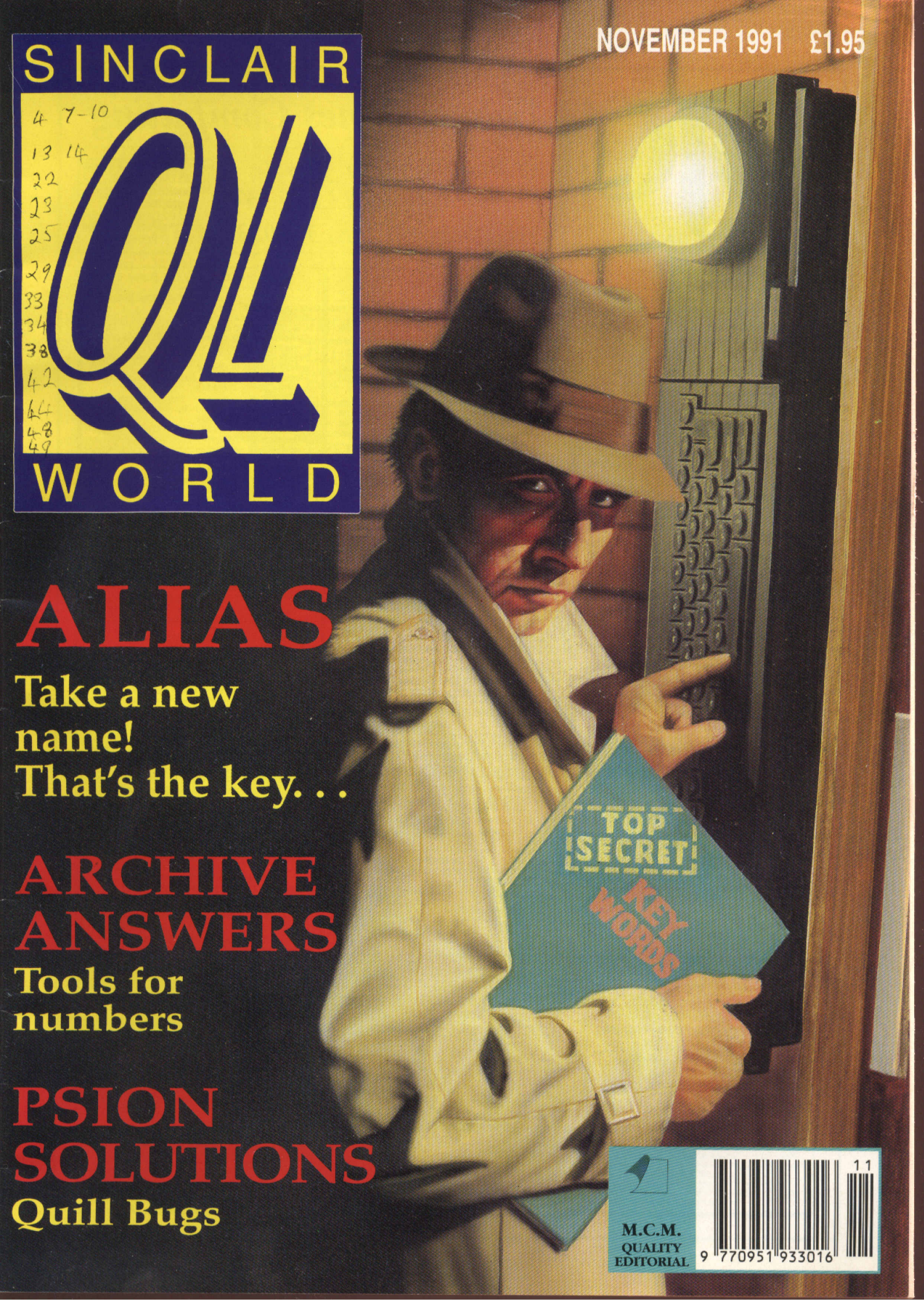
NOVEMBER 1991 £1.95

SINCLAIR QL WORLD

4 7-10
13 14
22
23
25
29
33
34
38
42
44
48
49



ALIAS
Take a new name! That's the key. . .

ARCHIVE ANSWERS
Tools for numbers

PSION SOLUTIONS
Quill Bugs

M.C.M. QUALITY EDITORIAL
9 770951 933016 11

SINCLAIR
QL WORLD

Editor
Helen Armstrong

Production Controller
Jayne Penfold

Designer
Jeff Gurney

Advertising Manager
Jason Newman

Magazine Services
Yvonne Taylor

Advertising Production
Michelle Evans

Group Advertising Manager
Lynda Elliott

Group Editor
John Taylor

Publishing Director
Wendy Palmer

Group Deputy Managing Director
Ray Lewis

Group Managing Director
Peter Welham

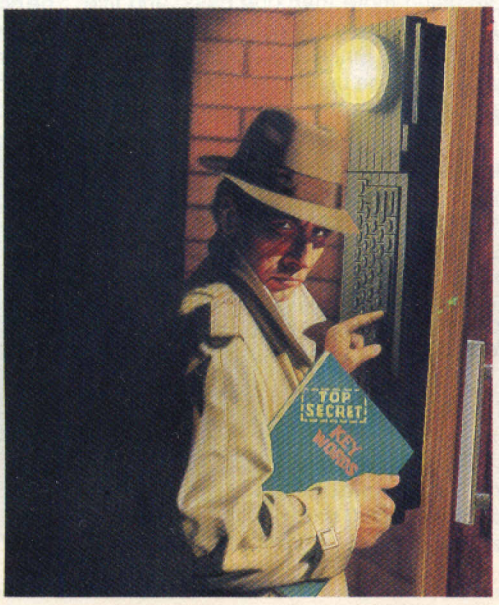
Sinclair QL World
Panini House
116-120 Goswell Road
London EC1V 7QD
Telephone 071-490 7161
ISSN 026806X

Unfortunately, we are no longer able to answer enquiries made by telephone. If you have any comments or difficulties, please write the The Editor, Open Channel, Trouble Shooter, or Psion Solutions. We will do our best to deal with your problem in the magazine, though we cannot guarantee individual replies.
Back issues are available from the publisher price £2 U.K., £2.75 Europe. Overseas rates on request.
Published by Maxwell Consumer Magazines, A Division of MCPC Ltd., Sinclair QL World is distributed by IPC Market force, King's Reach Tower, Stamford Street, London SE1 9LS.
Subscription information from: MCM Subscription Dept, Lazahold Ltd., PO Box 10, Roper St, Pallion Ind. Est., Sunderland SR4 4SN. Tel: 091 510 2290
UK: £21.00, Europe: £32.00,
Rest of the World: £38.00.
Please include your subscriber number with any queries.

Typesetting by Ford Graphics, 8-10 Whitsbury Road, Fordingbridge, Hampshire. SP6 1BR. Tel: (0425) 655657
Printed and bound by BPCC, Colchester. Covers printed by Spottiswoode Ballantyne, Colchester.
Sinclair QL World is published on the third Thursday preceding cover date.

CONTENTS

NOVEMBER 1991



NEXT MONTH

As well as our regular featurs, waiting for the off we have a review of PROSPERO PASCAL for programmers, PART 5 for followers of database DBQL, the second instalment of SYSTEMATIC MACHINE CODE PROGRAMMING, a ONE MAN'S SYSTEM about Archive, lots of reviews and some programs.

TROUBLE

A P R O B L E

Bryan Davies tries out *Perfection*, and Phil Borman's *SUB* routines

Digital Precision are pleased with the initial reaction to their new word-processing program, *Perfection*. There have been quite a few complimentary letters from proud new users. What criticism there has been appears, mostly, to relate to misunderstandings about the use of the program. As with almost any program, there will be many second thoughts after the initial release, some resulting from user feedback, others from the producers' own experiences during long-term use of the program. It is something of a truism in the micro world that no program is fully functional until version 3, but Perfection works well at version 2.

One thing that confused me was the mechanism for saving SuperBasic files. There is no problem loading them – just use the normal Load command – but the fact that they are 'plain text' means you cannot use the Save command if you want the SB program to run. Using the Export File command is the answer, but it may not appear to be so in some circumstances. If changes are made to the file, but the ENTER key is not used during the making of them, all should be alright. However, use of the ENTER key introduces a code into the file which will prevent subsequent running of the SB program. You can see if codes are present by using CTRL-H to display them. The remedy is simple – switch the Line Wrap function off before loading. Additional comment on this is likely to be put into updates to the instructions.

Cut and Paste

One function which seems to be missing from the QL utility scene is a routine for doing Cut and Paste operations on text, from program to program. For example, copying a database record complete with format from, say, *Archive* to *text*^{87} while both programs are loaded in the QL, during the same session. *Flashback* can do the paste part, but it really needs a program which 'sits above' the other programs to be able to both cut and paste. This is done in various forms on the PC and could presumably be implemented on the QL too. The 'ideal' form is where the two programs are linked, so that the required section is actually copied from program to program, and any alteration in the original data is reflected in the copied data, in the other program. This function has come to be called DDE (Dynamic Data Exchange) and must be something many users have dreamed of. A much simpler, but generally adequate, method is to copy the screen and transfer that. This works well when the screen contains text, but is not good for graphics. The mechanism is to make the source program the current job, activate a Mark function, and use the cursor – now tied to the supervisor program – to mark an area of text, by pressing ENTER at top left, then bottom right. You then switch to the target program, call up the supervisor program, and direct it to transfer the marked screen area. The text from the marked area then appears at the cursor point, and is inserted character by character, rather like being typed in; it looks very much the same as entering text from an ALTKEY, KeyDefine, or such.

My problem with printing on a laser printer from *Professional Publisher* now seems to have vanished, without the reason being apparent. Printing a full page from 1- to 3-pass, is now successful; the output is no longer spread over several pages. One possible reason is that the printer may previously have been set to Landscape instead of Portrait. On the laser, there doesn't appear to be anything to be gained from using more than one pass, and five passes still cause a print to be spread over two sheets. Since the FX80-emulation mode has to be used, the image quality is generally much the same as with a dot-matrix printer, although it should be appreciably better on areas of solid black.

While on the subject of printing, another improvement in my screen dumps has come about through a little more reading of the Trump Card instructions. The Epson GQ-5000 laser can emulate the LQ-2500 24-pin dmp, and it seemed reasonable to try the toolkit command <SDP_SET 6,3,0,0> for selecting the printer type and resolution. The resolution for these settings is 180 dots/inch (maximum for the laser is 300) and 180 lines/inch. The improvement in screen dumps was quite marked. The horizontal white lines we normally have to put up with in all-black areas are gone (see the illustration of a Perfection screen). Apart from using the LQ-emulation mode, the printer was set to print Landscape, to avoid the edges being chopped off the image. How applicable these setting are to other users' printers is another matter. If you have a GQ-3500 or -5000, or an actual LQ-2500, or something that can – or might – emulate it, it's worth a try. The dumps can take a long time though. The previous best settings for my 9-pin dmp were <SDP_SET 1,3,0,0>, which are for the Epson MX80 'or similar'; the resolution for this is 120 dots/inch and 72 lines/inch.

SUB for hard disk

The collection of utility routines for hard disk assembled under the heading of SUB by Phil Borman of Quanta is proving very useful with the Miracle hard disk. It would be no exaggeration to say they have transformed operations with the drive. Several programs can now be run, from their own sub-directories, using a further level of sub-directories for their data files. Switch-

A sample screen from *Perfection*.

ing between them involves no more than using CTRL-C – there is no need to re-acquaint each program with its default directories. Memory permitting, it is possible to run *text*[87], *The Editor*, *Quill* and *Perfection* together. With text[87] and Perfection, there appeared at first to be no problem, since they can both be configured to accept device names which are long enough to allow for sub-directories. In their cases, the program directories used are WIN1_TXT87_ and WIN1_PERF_, and the data directories are WIN1_TXT87_DOCS_ and WIN1_PERF_DOCS_. The abbreviated names are used because sub-directory names are actually only file names and there can be some confusion between program files and sub-directories if the latter are called 'TEXT87' etc.

Accomodating

In practice, it was found that text[87] would accept its directory names, but proceeded to delete the characters between the last two underscores and then could not find its data files. The other two programs are less accommodating, perhaps because they were written before hard disk was much talked about on the QL scene. As long device names are unacceptable to the configuration routines of *Quill* and *The Editor*, the default program devices were set as SUB7_ and SUB5_ respectively, and the data devices as SUB8_ and SUB6_. In view of the slight problem experienced withtext[87], it also was treated in the same way, the defaults being set to SUB3_ and SUB4_. The SUB routine itself translates SUB1_ to SUB8_ into specified sub-directory names.

In this case:

SUB1_ is equivalent to WIN1_PERF_
SUB2_ is equivalent to WIN1_PERF_DOCS_
SUB3_ is equivalent to WIN1_TXT87_
SUB4_ is equivalent to WIN1_TXT87_DOCS_
SUB5_ is equivalent to WIN1_EDITOR_
SUB6_ is equivalent to WIN1_EDITOR_DOCS_
SUB7_ is equivalent to WIN1_QU_
SUB8_ is equivalent to WIN1_QU_DOCS_

These equivalences could be produced by making the statements:

```
SUB_USE 1, 'PERF_'
SUB_USE 2, 'PERF_DOCS_'
SUB_USE 3, 'TXT87_'
SUB_USE 4, 'TXT87_DOCS_'
SUB_USE 5, 'EDITOR_'
SUB_USE 6, 'EDITOR_DOCS_'
SUB_USE 7, 'QU_'
SUB_USE 8, 'QU_DOCS_'
```

but the SUB routine makes it unnecessary to issue these statements. The program file SUB_BIN can be configured to be aware of what SUB1_SUB8 represent and, from there on, the user doesn't have to bother about typing-in the sub-directory names for each program. This may sound complicated, but it is much preferable to not being able to use the hard disk as a proper sub-directory device. Trying to sort out which files belonged to which program when they were all dumped together in the Root Directory was a real pain.

There are several useful commands and programs provided in the SUB collection, but it is available only to Quanta group members at present. Although I've not checked it yet, all the foregoing comments on SUB should apply equally to the high-density disks with the *Gold Card* as they have the same sub-directory structure.

The following report comes indirectly from two other users, and is not something I have been able to check yet. It is said that, when a system has both the Gold Card and the Miracle hard disk connected, the file WIN_REXT *must* be deleted from the hard disk. Miracle advise only that the file *is not necessary*, but that advice would appear not to be strong enough, as there is a risk of all hard disk files becoming read-only if that particular file is present during boot-up. The file is incorporated into the Gold Card rom.

There is an oddity in the instructions with the Gold Card, on the second sheet, where the command <FORMAT "MIRACLE*H"> is given an explanation with some surplus words in it. It will actually format a disk to high density (1.44 MB), with the label "MIRACLE".

Hardware gaps

Various Quanta members are active in trying to plug QL hardware gaps. The group itself is producing the Qimi mouse interface (as designed by QJump), and an IDE interface for hard disk is under development by one member. The Rebel hard disk units look as though they may reappear also. The interface part of the latter would be of particular interest to people who have, or can get hold of, cheap hard drives with the standard ST506 interface, because it should be possible to connect them directly to the Rebel interface. Some improvements have been made to the drive software recently; from accounts of users, the drive access speed with the Rebel unit is higher than that of the Miracle unit. The IDE interface should provide much better data-transfer rates than we have seen so far, but it does not support standard ST506 drives. There are a fair number of IDE drives available on the PC market at reasonable prices, though.

With the Qimi mouse now seeming to have no serious competition, could it become the 'standard' for the QL? There must be a fair number of them around, from the previous sales activities of QJump, Care and Jochen Merz. One obvious problem is that *Turbo*-compiled programs are apparently not compatible with Qimi. This would mean that *Professional Publisher*, for instance, could not utilise the Qimi mouse properly. This is perhaps the most important program in need of a standard mouse interface; it can be used with the Smiling Mouse, but there are not many of them around. It will also be important for Qimi and the Gold Card to be fully compatible; one would expect Qimi owners to be likely buyers of both the Gold Card and Professional Publisher. Miracle have recently made changes to the Gold Card rom to avoid possible trouble with Qimi interface. The borrowed GC I am using at the moment works alright in two of my QLs but not in the owner's QL with Qimi installed; there have been problems which appeared to be traceable to the Qimi installation previously, so some further experimentation is called for, and a later rom may be fitted to the GC shortly also.

It was encouraging to find that a particular JS QL, which locks up within about 20 minutes when a Trump Card is installed, doesn't seem to be having any trouble with the Gold Card, during all-day sessions. How touchy some of our machines are... It would suit me if the JS could be put back into use again (it went into a drawer once the Trump Card made it unusable), as the JM that has been used since then causes

quite some extra work during program-review sessions. There are a few features of the JM which don't matter in normal use but which bring work to a halt fairly often when new programs are loaded after the system boot has been run. It may be that some of the stoppages are caused more by careless programming than by the Qdos version. One regular fault is for a programmer to use a runtime Turbo toolkit without bothering to make sure the version of it that is supplied is compatible with the version of Turbo used during compilation of the program.

The chances are that a program was developed on a JS (JSU if written in the USA), and never checked on a JM, and something that can be got away with on the former causes a halt on the latter. Try this boot for size – can you spot why it doesn't run on a JM with Trump Card but runs without murmur on a JS with Gold Card?

```
WINDOW #0,480,23,15,229:BORDER
#0,1,7:PRINT #0,">>> One moment...":
SDP_KEY'p':SDP_DEVSER1:SDP_SET
6,3,0,0:
_base=RESPR(5632):LBYTES
flp1_RUNTIME_EXTS,_base:CALL
_base:
PAUSE100:CLS#0:PRINT#0,">>>Load-
ing 3D_TERRAIN...":
LINK_LOAD "PLOT","PARAMETERS",
"SHAPES","LIBRARY":END_CMD
```

In the boot file, this is actually one endless line of commands, with everything – apart from what is shown above as the second line – being in the original boot file. In the original form, the boot runs on both JS and JM, but the apparently-innocuous set of three commands to permit dumping of the screen cause the message 'bad name' to appear when the modified boot is run on a JM; on a JS, it still runs normally. Using the NEW command after the runtime extensions have been called, then typing-in the remaining commands, solves the problem.

Suppliers' letters

Note the change, for this month only. As I haven't had my next batch of readers' queries yet, but a bundle of letters has arrived direct from DP, it seems sensible to report a few comments from the other side of the fence. Summed up, what DP is saying is: 'Why don't customers check what they did when they placed an order, before complaining about non-delivery of the goods?' The mistakes made by customers when placing orders look easy to spot – afterwards – but we all have our blind spots when re-reading what we have written:

1) Incorrect supplier address. In DP's case, the obvious mistake is putting "22" instead of "222" in the street address.
2) Failing to put your own address on letter or envelope.
3) Not giving *all* the digits of a credit card number.
4) Not specifying the size and format of disk a program should be supplied on.
5) Calling and not leaving a message on the answerphone (then complaining that nobody called back!).

It's worth noting here too that the *QL World* editorial office often receives enquiries, changes of address or even cheques from subscribers who don't quote their subscriber numbers, and which would in any case have been better sent directly to QLW's subscription agent (see the small print on page 3).

If he has not written again, perhaps the customer who wrote to DP from Spain on 9th July about a problem using a 24-pin printer with *Professional Publisher* would contact them again, and give his name and address this time. In regard to his printing problem, it may be caused by his new Epson printer not being compatible with his old Epson one; specifically, it may use increments of 1/180 inch instead of 1/216 inch when up-spacing. The measurements of his two samples show the 1.2:1.0 line spacing one would expect from this, although the actual characters are larger too. Don't expect all Epson printers to be 'Epson compatible'. My Epson laser printer has two emulation modes for other Epson printers, as well as its 'native' mode, because Epson chose not to make newer printers completely compatible with the old FX80. For instance, the LQ-2500 can print in 9- or 24-pin mode, and it may be compatible with the FX80 in 9-pin mode, but it certainly isn't fully compatible in 24-pin mode.

SOFTWARE FILE

SOLITAIRE

INFORMATION
Program: *Solitaire* V 1.01
Price: £15.95 (includes air mail charges, but check current price before ordering. Visa and Master Card accepted).
Supplier: Sharp's, Box 326, Mechanicsville, Virginia 23111, USA. Tel: (USA) (804) 730 9697. Fax: (USA) (804) 746 1978

Bryan Davies takes some time to be alone with his computer.

Solitaire is a standard game for computers, but it doesn't seem to have attracted as much attention on the QL as some other games, such as *Othello*. Maybe the problem of making acceptable images of playing cards puts programmers off. The writer of Solitaire has made a commendable job of the cards, and their 'pips'. Lack of a standard QL mouse makes it necessary to write such games for keyboard use, or restrict them to the very limited market of one type of mouse, or write different versions for several types of mouse. The latter two options make no sense for a supplier with a small market, so this version is played from the keys. It is compiled with *QLiberator* and there are no serious delays in operation.

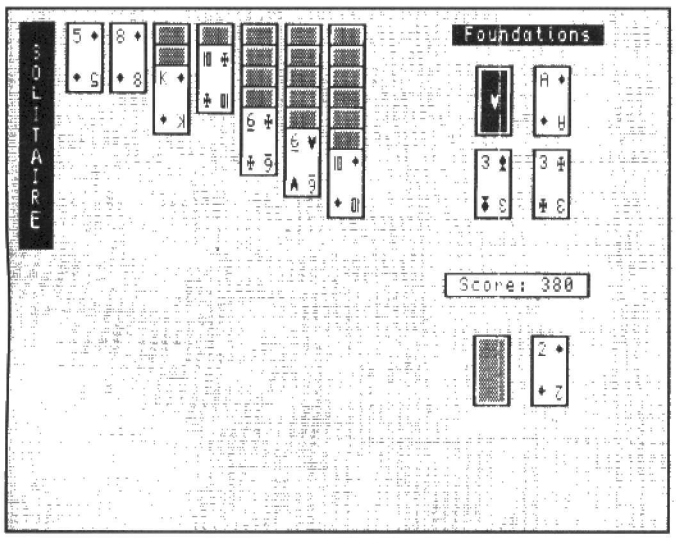
Instructions are in a Quill_DOC file, and are quite straightforward, taking up a little more than one page. The boot file serves only to put up the initial screen, and EXEC_W the program file. You can alter this to EXEC and multi-task the program, but there is only one point at which a cursor is displayed, so you need to be there when switching to other programs. The first actual program screen is a list of high scorers, to give you something to aim at. The lowest score on there is 150, the highest 1550 – quite a range. There are two levels of play, with the lower one being the easiest and scoring only half the points of the higher one. My first attempt netted 650, with the half-points level, so neither disgrace nor a prize looked likely.

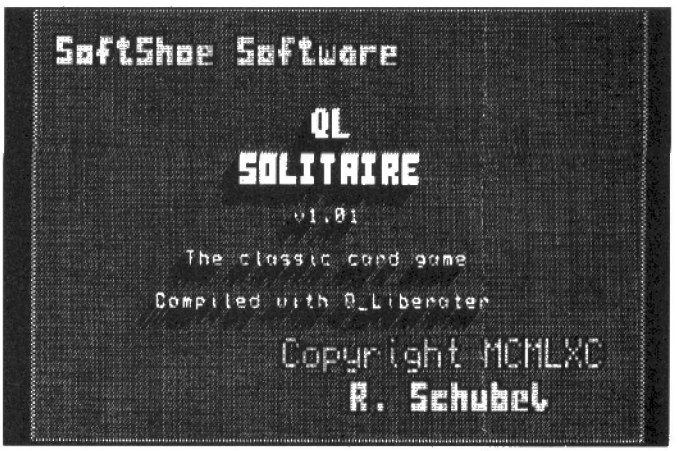
Once the game gets going you are aware of one of the QL's big weaknesses, namely the lack of colours in high-resolution mode; text is displayed in white on a green background, which is not a very visible combination on my Cub screen. Apart from this, the cards and their markings are well done, and the overall layout is clear. Ten keys are used to signal the moves you wish to make. You press a key to identify one of the seven columns into which the cards are initially dealt, one to move a card to the 'foundations' (the four piles onto which the suits are assembled by you), one to move a card from the deck (the remainder of the cards, not in the columns or foundations) and one to turn cards from the deck three at a time (only one at a time for Level 1). The program appears to trap 'illegal' moves (but no attempt was made to try all the possibilities for this).

Games tend to leave me totally cold, especially if they concentrate on eating or acquiring treasure of some form, but this one interested me enough to want to play it again and again. Maybe it is a function of simplicity, or some Luddite streak in me. Whatever the reason, Solitaire kept me at it for several dozen tries, after which my score had finally exceeded the highest on the default list by a sufficient margin for me to retire gracefully. It reminded me a bit of the 'battleships' game tested some while ago, which seemed to retain interest by virtue of its sheer simplicity, and some good touches on the display.

A very minor fault is that when y is selected in answer to the question 'Another game?', the y sometimes turns up as the choice for the next option, 'Please select level 1 or 2'. Since the y is ignored if you key in 1 or 2 and press ENTER, no disruption to the flow of the games need occur.

This is a good little game to have around for some light relief from serious computing. Reasonably-priced, it's easy to learn and play. You are unlikely to make the maximum score, so there's always a target to aim for. Not too taxing for us simple folk!





QL SCENE

Expansion board from Falkenburg

Jurgen Falkenburg (Computer Technik) has produced a new expansion board for the QL. Falkenburg reports that the *QL-ROM-Card* 'allows for the first time access to the complete QL memory range for several applications'. He recommends it for both unexpanded QLs and models with current ram expansion and disk drives. It can be used to expand rom or ram memory. With the additional power-protected *MOS-RAM-Disk,* it supports the installation of a reset and power-failsafe, write-protectable RAM_Disk (mos1_ to mos8_), with up to 256 KB for the QL with two *ROM-Cards* installed. Selection between different applications is done with an address range selector and other features of the expansion board.

Four memory banks are available on the board, each with a type-selector to choose between the sram and eprom available.

The board's integral battery backup and write protection switch all the use of static rams to replace eproms, so that an external eprom programmer is not necessary. The buffered rams are designed to hold their contents for more than six months before the batteries have to be recharged.

With the addition of the MOS-disk driver available, the MOS-Ram-Disk can be installed. This is recommended by Falkenburg as 'ideal for users working only with microdrives'. Frequent use programs can be copied to the ramdisk and can then be started from the MOS-Disk. Rommable software like Quill can be run directly from the MOS-Disk memory, leaving ram space free for data.

More than one ROM-Card may be used simultaneously, depending on the number of other peripherals in use with the QL. The free selection allowed from each memory socket allows a variety of combinations which are explained in the user manual. Basically, the different memory types may be expanded with a QL-ROM-Card depending on base address selected and the type and size of memory installed. The QL may be expanded to 1008K of memory with a maximum of 368 K rom, 896 K ram and 256 K MOS-Disk.

The QL-ROM-Card cannot be used alongside the TrumpCard with 738 K ram, or the Gold Card.

The export price of the QL-ROM-Card is DM 157 (£52) or DM 210 (£70) including the MOS-Disk driver and static rams.

The fact sheet from which this information is drawn is available from **Jurgen Falkenburg, Thanweg 36, D-7359 Ersingen, Germany.**

MORE ON QIMI

Quanta has contacted QL World with further information about the *Qimi* mouse and interface *(QL Scene,* September 1991). Manufacture of the Qimi interface board in the UK has been commissioned by Quanta, who are therefore able to offer the board (to Quanta members only) free of commercial overheads at the considerably reduced price of £25 plus £2 post and packing.

Bill Richardson of EEC has told Quanta that he will no longer be selling the imported interface board, but still has a stock of Qimi Mice available for sale. The UK-manufactured board will be available to non-members from Data Systems for £35 plus £4 post and packing.

Says Quanta Chairman Phil Borman: 'Some of our members misunderstood the report and assumed that Quanta must be buying its boards from Jochen Merz. This is not the case.'

Data Systems man Chris Gregory can be contacted on 0272 513653 during social hours.

Changes in Spain

The Spanish QL users' club in Madrid is now named *QLiper,* and publishes a 3.5 inch disk-based magazine bi-monthly. QLiper welcomes any owner of a computer running a Qdos compatible operating system.

Contact Marcos Cruz, Acacias 44, 28023 Madrid, Spain.

All Formats dates

Dates booked for the All Formats Computer Fair for the remainder of the year are: London Horticultural Halls, Westminster, November 3; Midlands, National Motorcycle Museum, opposite the NEC, November 10; Scotland, City Hall, Candleriggs, Glasgow, December 1; London Horticultural Halls, Westminster, December 14; Leeds, University Sports Centre, December 15. The organisers write: 'Tip: Come after lunch, when there are no queues, stallholders are more accessible and prices are often even cheaper.'

Advance tickets and information from **John Riding, tel: 0225 868100.**

Toolkit errata

Two printing errors appeared in DIY Toolkit in last month, for which we apologise.

The listing one on page 38, one line is missing from the bottom of the listing. This should read:

bne.s check_chan

The seventh paragraph in the first column of page 39 should read: 'The Turbo-Toolkit variant has the most comprehensive checks. Listing two detects most errors, but may stop if you run out of memory. . .'

OPEN CHANNEL

Open Channel is where you have the opportunity to voice your opinions in *Sinclair QL World*. Whether you want to ask for help with a technical problem, provide somebody with the answer, or just sound off about something which bothers you, write to: Open Channel, Sinclair QL World, 116/120 Goswell Road, London EC1V 7QD.

Sorcerer

I was recently shown a copy of *Sinclair QL World* by a friend of my daughter. He owns a QL (several, in fact) and is currently working on a very high resolution graphics board for the QL with a lot of success.

The copy in question was the May 1990 issue, and contained a Printer Report in which three contributors described their experiences when linking three different printers to the QL.

One of your contributors, John de Rivaz, tells of his efforts to link a NEC P2200 to his QL and to his Exidy Sorcerer, and of his need for some kind of reset feature.

I do not own a QL but I do own a Sorcerer (several, in fact) and have had similar problems linking it to an Epson MX80F/T.

Concerning his reset problem, I have fitted a neat looking pushbutton (single pole) to both my Epson and my Star. This pulls the INIT signal line (active low) at Centronics input pin no. 31 down to 0V ground when I want to clear the printer buffer of rubbish. This method does not cause the Sorcerer to crash as switching off the printer does.

I find his remarks about the Sorcerer and his word processor very interesting because I too had similar problems. The word processor program I use is one originally produced by Exidy and Testan Scientific together and supplied to Sorcerer users in a Rompac. I have a similar program on tape and on disk which I have modified to enable the wordprocessor program to control an Epson printer via codes embedded in the text. If his word processing program is similar to mine in having the output to the printer fed via a vector address, then he could insert this patch into the output chain quite easily by changing the vector address to point to the patch.

The patch I have is in Z80 code (a cpu that Sinclair long ago abandoned!) but the algorithm could be used on the QL so if anyone is interested I will reply to any comments or queries I get.

I am particularly curious about the set-up that Mr de Rivaz has with his Sorcerer and wordprocessor and would be grateful if you could put me in touch.

Jack Swain
Hitchin
Herts

Editor's Notebook

This month sees the end of the 'mainly for beginners' section of the *New User Guide*, and we know from the letters we have been receiving that new and not-so-new users have found the guide helpful in clarifying and supplementing the original Sinclair QL User Guide (pages 25 to 28).

But fear not! We continue next month with the second part, on SuperBasic and the QL keywords. Author Mike Lloyd compares the 'beginners' section to getting ready for a driving test. Now we can get out on the road.

As a footnote to comments by Digital Precision's Freddy Vachha in Troubleshooter this month, reader Grigoriadis Stathis wrote to us to ask advice on importing QL hardware to Greece. Our first hint is: give your full address. If he writes to us again with his full address, not only will we write to him, we will forward his enquiry to experienced exporters Miracle Systems. We also have a packet for Erling Jacobsen, thanks to fellow reader GM Pheasant, but not his address.

Please write, Erling.

Gold Award

Digital Precision Ltd. only praises when praise is due; consequently we don't praise other people's products very often! However . . .

Gold Card is fantastic, terrific and superb, and is both strongly recommended and endorsed by Digital Precision Ltd. It is excellently designed and engineered. It is quite a bit faster than its manufacturers claim (about eight times faster than a 'standard' QL and about five times faster than the most common Trumpcard). It is very reliable, and, perhaps most importantly, it is very compatible – compatibility with most programs (and all of ours, as far as we know) is perfect.

Miracle Systems are to be congratulated for Gold Card and castigated for the over-modesty of their advertisement for it: Gold Card is far better than they make out!

Many Perfection users have told us they think Perfection is the QL product of the year. Digital Precision disagrees – pride of place goes to Miracle for Gold Card (with Perfection a close second). Bravo, Miracle! Long live the QL.

Freddy Vachha
Managing Director
Digital Precision Ltd

Precision

I bought my first QL in January 1987 after reading a review in that month's edition of QL World of a new word processor by Digital Precision called *The Editor*. The description indicated that the program was efficient, fast and very flexible. So it proved to be, with further extensions in scope as it was developed into *The Editor Special Edition*.

Over a period of time, one becomes attached to a word processor as proficiency and familiarty increases. To contemplate change involves quite a wrench, but in May 1991 I again read in QL World a glowing report of a new word

processor by Digital Precision, this time called *Perfection*. Clearly much had happened in the period between the two programs, and I decided to purchase it.

The program has a precision feel to its operation, continues the tradition of flexibility and is very fast. It seems designed to give equal satisfaction to occasional users who can mainly implement commands via the menus and regular users who have the opportunity to use the faster commands.

I have no connection with Digital Precision and its design team, but I offer them my congratulations on a major development for the QL.

R E Copland
Ryhope
Sunderland SR2 0HT

QL and Mac

I am an ex-QL user (largely) but would like to suggest to QL owners and Amstrad that Amstrad make a cheap CE-1 computer that could run Mackintosh software and be QL-compatible.

Such a machine with a 68020 or 30 processor could be upgraded to 68040 in future for real QL power.

By the way, I have some QL software and blank cartridges. If any QL owners in Australia would like a list of titles, please write to me.

Wayne Mocrellini
PO Box 309
Gordonvale Q 4865
Australia

Registered

For years I've borne, with irritation, the fact that *Quill* and my printer have never reliably kept continuous pages in register. Now I have finally got the solution. First, the preamble in the printer data has to be extended using Install_bas to cancel the automatic perforation skip of the printer. On my Epson-style printer this is done by adding 27.79 (ESC, capital 0). Second, the lines per page in the printer data has to be changed to 70 for A4 paper (presumably 72 for 12 inch paper). Thirdly, using 'Design' on the Quill document, the upper and lower margins are set to zero and the lines per page to 61. The 70 is essential, since it fixes the pitch of the page. The upper margin is set by where you set the page in the printer before starting. The lower margin is set by the 61, since this number fixes how much of the page is printed, counting from the first printed line to the 'page n' which Quill prints three lines below the last line of text on the page. The number of lines actually printed per page is therefore 58. Choice of 61 gives a four-line clear margin at the foot of the page. This means that this number can be set to anything up to a maximum of 65.

None of the above information is obtainable from the original documentation, which specifically states that the default settings of both 'Design' and the printer data are for A4 paper. The 'page n' mentioned above, and the spacing between it and the text, is a default setting of the 'footer', a fact which also is not stated in the manual, and which it might take the casual baffled user, who never has use for a footer, a year to find out. The setting can be changed using the 'footer' instruction.

Walter Stanners
Over
Cambridge

Underlined

In the May edition of *QL World*, Beryl Crawley described a problem with *Quill* which had the effect of underlining all her screen output. I also had a similar problem, but it disappeared some time ago and I am not sure why. I suspect it might have to do with the QL overheating, in which case I recommend Care Electronics' QPower Regulator, which I found simple to fit (I am not an engineer). The only other 'modification' I have carried out on the QL is to buy a mains interference suppressor, available from most electrical shops.

G W J Daniel
Heath
Cardiff

Basics

QL users are staunch supporters of Qdos and SuperBasic. The operating system is without doubt an excellent one, but though SuperBasic is powerful there remain possible enhancements which can be found in other Basics. There are times when I look enviously at the Basic of Acorn computers, which have such things as VAL and OLD, or for that matter CHAIN, which allows one Basic program to call another and pass parameters to it.

I also like some of the Basic constructs of the SBASAG extension to Research Machines' RMBasic, written for the RM480Z machine by Software Production Associates. It is not difficult to utilise SuperBasic keywords to emulate all of these, indeed, LOOP. . . ENDLOOP/REPeat. . . END REPeat and CASE. . .ENDCASE/SELect. . . ON. . . END SELect are equivalents, but the availablility of the WHILE. . . DO. . . ENDWHILE and IF. . . THEN. . . ELIF. . . ELSE. . . ENDIF constructs would be an enhancement to SuperB as both would improve the readability of programs by avoiding the need for additional nested conditional IF. . . END IF or SELect. . . END SELect constructs.

I particularly like ELIF, which would read better in its unabbreviated form of ELSEIF or ELseLF. For those unfamiliar with SBASAG, the specimen here shows the use of the SBASAG keyword ELIF.

Much neater, don't you agree? Perhaps some enterprising reader could, in the interests of portablility of software and of general flexibility, write some SuperBasic extensions to emulate these to save the chore of amending such software. For the RM480Z machine, SuperBasic's EDIT, CLS with parameter and WINDOW would also be welcome USR additions.

Other readers may know of other desirable extensions to SuperBasic that exist on other Basics. Simon Goodwin occasionally requests suggestions for toolkit extensions. I wonder if he, or any other contributor, would consider writing some articles showing alternative ways in which the construction that exist in other Basics could be emulated on the QL using existing keywords, and also write some machine code extensions for the more useful ones.

Perhaps you would consider commissioning a series in the format of the old monthly Keyword page. This would serve the purpose of enhancing our own Basic and making it easier to port Basic programs written on other computers over for use on the QL. I recall that this was the justification for having some of the interior Basic constructions such as the FOR. . . NEXT loops, GOTO and GOSUB in the QL rom. We still have these, but wouldn't it be even better to have their extras also?

Ron Allpress
Thwaite
Suffolk

```
IF <expression> THEN
    <statements>
  ELSE
    IF <expression> THEN
        <statements>
      ELSE
        IF <expression> THEN
            <statements>
          ELSE
              <statements>
        ENDIF
    ENDIF
ENDIF
```

```
IF <expression> THEN
    <statements>
  ELIF <expression> THEN
    <statements>
  ELIF <expression> THEN
    <statements>
  ELSE
    <statements>
ENDIF
```

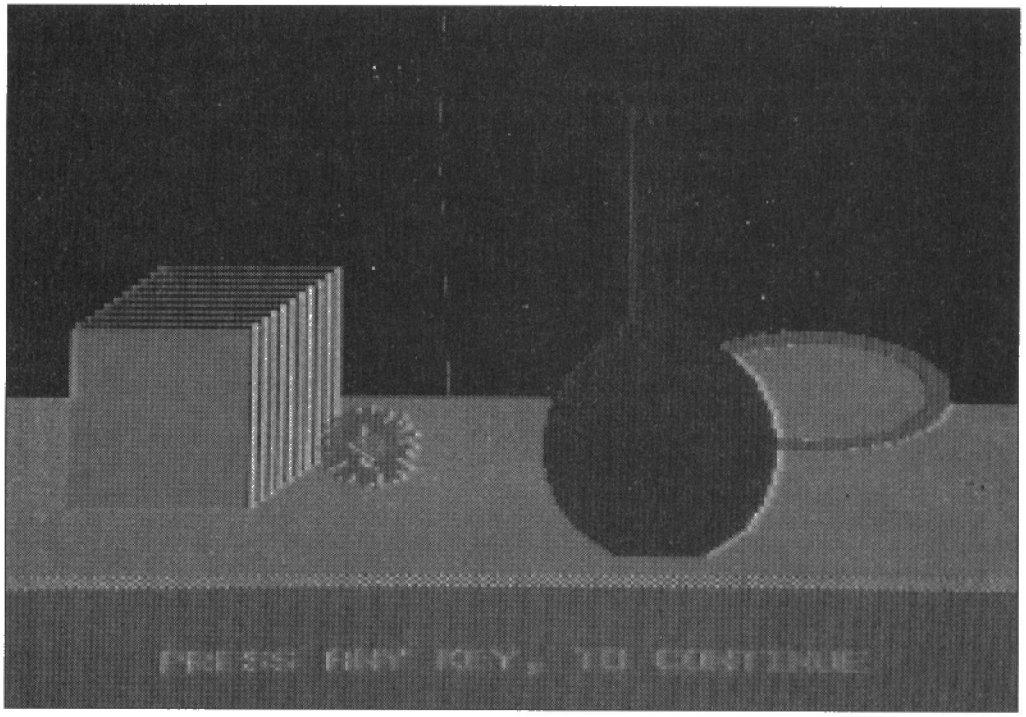
SOFTWARE FILE

INFORMATION

Program: *Vision Mixer Plus.* Needs min. 256K ram, and monitor display.

Supplier: Dilwyn Jones Computing, 41 Bro Emrys, Tal-y-Bont, Bangor, Gwynedd, LL57 3YT.

Price: £22.50. 3½ or 5¼in disks only.



VISION MIXER PLUS

Vision Mixer Plus is a natural improvement on the video effects program *Vision Mixer* by Dilwyn Jones, which I reviewed in *QL World*, January 1991. The enhancements now include 50 wipe effects (many new), the number of screens is no longer limited by the available ram, mode 4 and mode 8 can be mixed and a few other small benefits. The main applications, as before, are seen to be in lecturing, business presentations, advertising and entertainment. Indeed they have been used very successfully at Quanta workshops, notably in Portishead.

The program comes with an easy-to-follow 16-page manual which contains all the information needed. A special section is added to show how the user can run from a hard disk setup.

This upgraded program allows the construction of more screens and sequences than was possible with *Vision Mixer* and requires only 128K of memory expansion. This is because *Vision Mixer Plus* does not require the whole sequence of screens to be pre-loaded into the computer's memory. Only one screen at a time is loaded just before it is due to be displayed. Two key characteristics follow directly from this:

John Shaw mixes it . . .

Firstly, the maximum number of screens which can be placed in a display sequence is not a function of the size of memory, but of disk drive capacity – 44 screens for a twin 3½inch drive using 1440 sector disks, 88 for four such drives (22 per disk), many more for a hard disk.

Secondly, the medium on which the screen images are stored must be available for constant access when the display sequence is running. Floppy disks holding the screen images must remain in one or more of drives flp1_, flp2_, flp3_, flp4_ (or the screens must be stored in Ram2_). A hard disk directory Win2_ holding such screens must be similarly accessible.

The peak memory required by the suite is about 100K, leaving a similar amount free even with only 128K of memory expansion. The main program MENU shows the free memory, updated each time the MENU is displayed. The program now permits very long sequences of screen images to be presented with, or without human intervention.

It includes a selection of effects for use in the transition from one screen to the next (wipes). In the random WIPE and random colour mode, the program not only offers many hundreds of different inter-screen effects, but it also provides for the WIPE colour to be fixed and/or specific WIPEs to introduce specific screens.

Users with larger memory expansions have the option of copying some, or all of the screens of a sequence onto a ram disk (RAM2_) before running the program (subject to available free memory). This facility can be used to supplement disk storage capacity, or to prevent excessive drive wear when running the program for long periods on a regular basis. A program, RAMLOAD_EXE is included on the disk to help facilitate this.

The time interval for the display of each screen is at the user's choice and can be very easily changed, as can be the choice of screens in a sequence, the order of their presentation, the allocation of WIPEs to particular screens and the WIPE colour.

Examples of some of the 110 wipes which are now included on the disk are:

1 Venetian Blind Down
2 Venetian Blind Up etc.
3 Diagonally Split Blockout
4 Venetian Blind in Halves Down& Up
5 Blockout in Halves Down & Up

6 Multi-split Venetian Blind Up & Down
7 As Wipe 6, but solid blockout
8 Two-colour Venetian Blind Down & Up
9 Vertical Venetian Blind
10 Wipe 1 Plus Wipe 9 – two colours
11 Wavy Front Blockout
12 Two wide "snake" Blockout with gaps
13 Multiple Blocks with horizontal gaps
14 Multiple Blocks with vertical gaps
15 Wipe 14 plus Wipe 13
16 Wipe 8 plus Wipe 15
17 Vertical Venetian Blind – wide gaps
18 Wipe 9 plus Wipe 17
19 Four narrow "snake" Blockout with gaps
20 Two-colour Concentric Circle Blockout

All of these screen sequences, and indeed the other ninety, can be SAVED and re-LOADed automatically for immediate use on another occasion.

Users of the original *Vision Mixer* will have to change their methods when using *Vision Mixer Plus*. The much higher overall speed and screen sequence capacity of the latter coupled to its much smaller memory consumption and ease of making changes should allow the imagination to run riot.

Screen

The *Vision Mixer* screen images fill the whole screen, as do the wipes. In *Vision Mixer Plus* the bottom window (channel #0) is used to show the message, or screen title, forming an inherent part of the saved Screen and is not affected by the wipes. This deliberate difference stems from the need to have a set space and format for descriptive text when it is used for lectures, or advertisements.

So here we have the natural progression to *Vision Mixer*, a Rolls-Royce program with the versatility and scope needed to give a much more professional presentation.

A companion program, *Picturemaster*, by Joe Haftke, also available from Dilwyn Jones Computing, allows for the generation of many more new screen images, creating captions, shadowed lettering and drawings, and producing some intriguing screen manipulation effects.

INFORMATION
Program: *Picturemaster.*

Supplier: Dilwyn Jones Computing, 41 Bro Emrys, Tal-y-Bont, Bangor, Gwynedd, LL57 3YT.

Minimum 256K ram required. Monitor display preferable. $3^1/_2$ or $5^1/_4$in disks.

Price: £15.00

PICTURE MASTER

. . . and gets into the picture.

Screenmaster is a comprehensive utility designed to make screens for use with *Vision Master* or *Vision Master Plus*. These screen images can be 'pictures', or text, or a combination of the two.

Text can be produced in a variety of styles, and visuals can be generated using lines, rectangles, circles and ellipses. The figures can be moved, rotate and have their size varied without any limitations and in any QL colour in mode4 and mode8.

Menu

Included are 30 ready-made procedures for making different screens by a simple choice from a menu. Most employ the use of random numbers, resulting in a much larger selection of screens than the 30 basic designs would suggest. They are as follows:

1. Star 2. Tapes 3. In_sets 4. Twirl 5. Flying Sign 6. Logo 7. Spirals 8. Flowers 9. Fans 10. Vortex 11. Icons 12. Colours_4 13. Colours_8 14. Kite 15. Pumpkin 16. Clover 17. Collage 18. Dart 19. Ant 20. Tent 21. Cones 22. Off_sets 23. Kaleido 24. Fish 25. Printsize_4 26. Printsize_8 27. Bookmark 28. Desert 29. Ellipses 30. Objects.

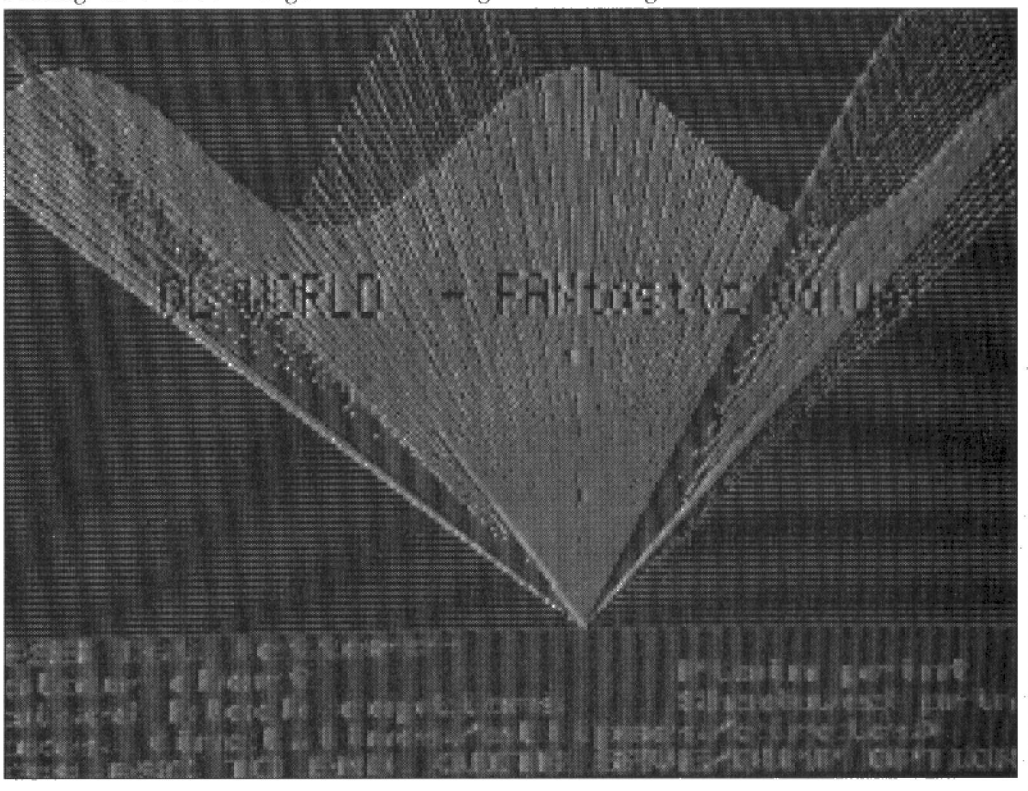
These and others can be dumped to a suitable printer, with the aid of the user's own copy of "gprint_prt" (a part of Psion *Easel*). The most important feature of the program is that it is deliberately made available in SuperBasic so that users can modify it.

You can generate a large number of pre-programmed, attractive screen pictures or adapt your own screens to the *Vision Mixer Plus* format with text captions or make text screens in a variety of colours, size and style combinations, very simply and fast.

Similarly, you can generate a variety of geometric figures and patters and create/enhance screen images, using any combination of the above.

If you prefer, you can QUIT the program and use its many PROCedures as direct commands. Used in this mode, the program still offers all the capabilities of the formal options, but allows the user the freedom to modify parts of the program, experiment and improvise.

The program is supplied with a comprehensive manual which is well laid out and easy to follow.



QL SCENE

WINBACK AND VISION MIXER UPGRADES

Dilwyn Jones Computing has upgraded its hard disk backup utility, *Winback*, as a result of the review in *QL World* September 1991. Some minor (and obscure) bugs have been fixed, and the problem associated with large files such as the MSDOS partition has been sorted out – the user now has the option of missing out this file and proceeding with the remainder of the backup. An upgrade to the program, which is now at version 1.10, can be obtained by sending the Winback master disk to DJC. The upgrade is free, but please include return postage.

The *Vision Mixer Plus* program has also been upgraded to make it compatible with the Miracle Systems *Gold Card* interface. Upgrades for this program are also available free of charge when the master disk is sent with return postage enclosed. The fix is required because the program previously crashed if the amount of free memory available was greater than 999,999 bytes (ie more than six digits long) and users had to resort to cutting the amount of free memory available with the RES SIZE command on the Gold Card toolkit.

Further information and returned masters to **Dilwyn Jones Computing, 41 Bro Emrys, Tal-y-Bont, Bangor, Gwynedd LL57 3YT, UK. Tel: 0248 354023.**

'Universal' Drives from EEC

Following the demise of MGT, EEC Ltd purchased the entire stock of 'lifetime' disk drives, for which they found a ready market, and they are now producing a 1MB and 2MB replacement to be known as the 'Universal' disk drive.

The drive, with the appropriate lead, is compatible with the BBC range of micro computers, the Archimedes, QL, Spectrum, CPC, Atari ST and the Amiga, as well as the PC XT and AT, and most PC compatibles including the Amstrad 1512 and 1640. It can be used with the Spectrum, using a Datel Plus D disk and printer interface, and with the QL using one of the Miracle expansion units, also available from EEC. The Universal drive is in a neat grey metal case with mains switch, lead and 13 amp plug. Selection for different computers is done by external dip switches, and the unit is supplied with a dust cover and comprehensive instruction and installation manual covering the listed micros.

The basic unit has an Amphenol-type socket on the back, similar to a Centronics printer socket, and leads are provided for the various options.

The price of The Universal disk drive is £75.00 inclusive of VAT and any one lead. Leads for the other micros are also available ex stock for £9.95. If a double drive is required, two units can be supplied for stacking side by side or on top of one another with a special lead for a total of £135.00 inclusive of VAT. Carriage in each case is £9.00, with delivery ex stock while stocks are available.

For further information contact **EEC Ltd. 18-21 Misbourne House, Chiltern Hill, Chalfont St. Peter, Bucks SL9 9UE. Tel. 0753 888866.**

SQLUG celebrate

It seems like only yesterday when Scottish QL Users Group SQLUG were celebrating their first year together. Actually, it was last September. Being as it has lately been this September – SQLUG are celebrating their second year together. To mark the event, alongside their monthly four-page SQLUG Newsletter, they have produced a *Second Anniversary Special Edition Newsletter*, with 16 pages (A5), an encouraging report of the year's activities, and two useful articles, a general one on upgrading QL systems, and another on repairing damaged disk sectors.

"We are keen to recruit new members, so we would be grateful if you could mention our continued support for QL users," writes Secretary Alan Pemberton. "For the £6 annual subscription, we offer help and advice at meetings or over the phone, a regular newsletter and access to our large public domain library (bigger even than Quanta's!) In fact, given all our benefits, I'm surprised we haven't attracted more QL users."

No sooner said than done, and many happy returns of the Special Edition. Contact SQLUG through Alan at **65 Lingerwood Road, Newtongrange, Midlothian EH22 4QQ.**

Quantem go International

Quanta's East Midlands subgroup, Quantem, have announced that their two-day workshop on the 12th and 13th October will be the first International QL Meeting in the UK. The meeting is being held at the Sherwood Community Centre in Nottingham, which was also the venue for the Quanta AGM. The venue was chosen for its central position in the UK and with regard to ports from Europe.

Personal Opinion

In the first of an occasional column, Eduardo Fairbanks, a long-time user and recent subscriber to QL World, puts forward his hope for the QL in the futiure.

The fact that the QL has not been produced since 1985 only reduced the strength of the QL scene a little instead of killing it. This is for two main reasons: first, because the QL and its Qdos are very powerful and friendly; second, because QL users and suppliers made a great effort to support the QL. Personally, I think that its long life is due to the genius of Sir Clive Sinclair, Tony Tebby and a few other brilliant people: thank goodness for having them.

Development

Reading specialized magazines we can find people who continue to develop hardware, peripherals and software for QL, and a number of them seem to be very good.

I only recently got a Quanta membership and a *QL World* subscription, but I have been reading about Sinclair micros for a long time. I have been a Spectrum user since 1982 and I have a very good system with eight networked Spectrums and all sort of peripherals. What always catches my eye in the magazines are the differences between European and American users' desires, both for the QL and the Spectrum.

On the subject of software most British users prefer word processors and spell checkers, desk-top publishing, tax accounts programs, and so on. American ones prefer to develop graphics systems, cad and drafting programs, and robotics.

On the other hand, Europeans early on developed interfaces for printers, joysticks and midi. Due to their software preferences Americans buy mice, massive memory cards, I/O and A/D interfaces.

Differences

I remember that in the earlier age of the Sinclair ZX80 and ZX81, one could already find in drugstores I/O interfaces to drive robotic models, speech synthesisers and speech recognition units. They also had disk drives for the ZX81!

Turning back to the aim of this text, I disagree with some people who have written that there are few things more to do on QL developments, now that it already has all a user could want. I recently read about some upgrades for QL users: in Italy Mr Innocenti has developed QL graphics so that he can put up to 32 real colours in mode 4. Mr Delsanti wrote a routine to animate screens and gave it the significant name of QLFilm. I know that some people are now working on hidden line and shading routines for cad packages. We have cad packages for IBM PCs.

Still in Italy there is Mr. Del Bello's say_italian routine running into a speech synthesiser, and Mr Santachiara's music manager. I recently heard about the new video digitizer on the market, from CL systems. Some other articles tell about expansion boards, real time clocks, robot control and so forth.

Increasing

As we can see, the list of new QL features is increasing much faster than one could think for a micro which hasn't been produced for a long time. We can also see that the differences between British and overseas users I pointed out in the beginning is decreasing.

The generality and the versatility we are now going towards necessitate the development of motherboards to multiply the expansion port and the eprom connector, as well as the improvement of the internal voltage regulator and heat sink in order to attach more interfaces. By the way, I already had that problem with my Spectrum system; today my eight Spectrums are powered by four IBM switched power supplies (with fans) giving them the right voltage and current direct to their pcbs, so I took out their heat sinks and voltage regulators, and they run at room temperature.

It is not a good thing to be powering the computer on and off to change peripherals every time you change jobs. It would be wonderful to have them all connected at once. Best would be an expansion board capable of switching each one or several on or off at our wishes without the need to unplug them. A chip switch should be provided to avoid system crashes when powering the peripherals on and off. The chip switch should be triggered both mechanically and from the software.

The dream

For myself I would connect the Trump card, trackerball, scanner, dot matrix printer, plotter, speech synth, modem, eprom programmer and whatever else is forthcoming. At that level of the dream, one could also design several switchable eprom ports!

Of course, this QL would have a lot of memory, like the old ABC Elektronic's 4 megabyte ram card, or even more. We know that the 68008 can address up to 16 MB of memory, and the Japanese chips are growing faster in capacity while falling down in price. So I think that very soon we could have a 16 MB ram board. An example of what I say is (once more) my Spectrums: I had one of them upgraded by VideoVault from 48 K to 128 K (people always said that could not be done, but it was) and I also have another Spectrum with a co-processor (4 MHz) and 512 KB dram! As we can see, things go on and on and almost anything is possible with electronics and creativity.

Another leap

I have been reading some articles about how we could upgrade the QL to perform another leap on it. Some people wish to have more memory, others more speed, others want to have more graphics resolution on screen, like the Atari and Amiga do. I really don't know how to improve the screen resolution, but a good look into the Macintosh circuitry would give a good idea. The speed is easy; we know that the 68008 runs by four steps of eight bits to perform its 32 nominal bits due to the economy of eight tracks instead of 32. Nowadays the price of a pcb containing 32 or 8 tracks is the same. One could also use

the 68030 or even the 68040, much faster than ours.

Do something!

Some people write articles upon this subject and call their project one name or another. Names apart, all of them want to upgrade their systems to the Quantam Leap Mark II because they love the QL and Qdos, and are not happy working with Atari, Amiga and IBM sets. When they have to do this, they use on these machines a QL emulator. Those people really want to do something in the near future, and I'm of a like mind.

Nevertheless some other people advise against this type of project on two counts: firstly, Amstrad's rights over Sinclair and the Quantam Leap; secondly, cost.

Well, the QL Mark II would have a rom like Minerva and an enhanced Qdos. Some people have written (and they are right) that the project would cost a lot in planning, design, production and marketing. They say that if established people and corporations in the market don't have interest in producing it, how can a group of users do it from the bottom?

Well, I have some ideas on how: great corporations and users have not exactly the same targets; they have profits to care for (that is their business), and we only want a product that fits well. But we have something that they cannot allow for: there are a lot of us.

We could combine our purposes and share the costs of the enterprises. I heard that in Quanta alone there are 2,000 users, still growing.

I also think that Spectrum users should be involved too; a lot of them would love to upgrade to QL. We could collect hardware projects that individual users had developed, and mix them to produce the major design. Of course main suppliers should be invited to combine with the whole group, sharing (selling?) experiences and technology, and supplying parts. Their quotes must have a different value and return rates, because of their financial needs.

Available

The design should employ parts already available in the market, like power supplies, keyboards, video encoders and so forth, to reduce the costs of small scale production of new parts. The project also must be concieved with the possibility of extending the equipment so people could afford some modules at a time.

Then a new worldwide enquiry would have to be carried out, to tell people what the new Mark II will be, what the versions would cost, and asking for membership payment.

I wish I could give some ideas to improve the QL scene, and I'd like to hear about others' improvements on that.

Eduardo Fairbanks
Brazil

If you have an informed view of how the QL has developed during its history and where it should go from here, and want to offer it for publication, please write to the Editor for an information sheet, or submit a double-line-spaced text of no more than 2000 words if you prefer. Personal Opinions should concentrate on the development of the QL in the light of personal experience. Responses to a published Personal Opinion intended specifically for the author will be forwarded; please say whether or not you are willing to have parts of your response quoted in Open Channel. Sorry, we cannot forward confidential mail. Short letters to QL World in response to Personal Options will be considered for Open Channel in the normal way.

aCcLcesBs

BELGIUM

Club Sinclair BruQsL (Belgium) Contact: Jaques Tasset, Aarlenstraat 104, 1040 Brussels, Belgium

SWEDEN

International QL Conference bulletin board system (Swedish and English). Contact: Michael Cronsten, System Operator, Jamten-TCL, SSoere 1073, 83030 Lit, Sweden

USA

New England Sinclair QL User Group (USA) Membership Secretary: Sherm Waterman, PO Box 8763, Boston, MA 02114 8763, USA. Magazine: *NESQLUG News*. Editor: Peter Hale, 195 Central Ave., Chelsea, MA 02150, USA.

NORWAY

Norwegian All Sinclair Association (NASA) Contact: P Monstad, NASA, N-5580 Oelen, Norway. Magazine: *Sinclair Magazine*.

ITALY

Qitaly Club Chairman: Roberto Orlandi, Via Brescia 26, 25039 Travagliato (BS), Italy. Tel. (local) +39 30 6863311. Magazine: *Qitaly Magazine*. Editor: Dr Eros Forenzi, Via Valeriana 44, 23010 Berbenno (SO), Italy. Tel. (local) +39 342 492323.

TURKEY

QL Qlub (Turkey). Contact: Bulent Artuz, Prof. Sitesi B/1 D/5, Etiler 80600, Istanbul, Turkey.

ENGLAND

Quanta (UK) Membership Secretary, Bill Newell, 213 Manor Road, Benfleet, Essex SS7 4JD. Magazine: *Quanta*. Editor: Bill Fuggle, 20 Widnes Avenue, Selly Oak, Birmingham B29 6QE.

Bristol sub-group: Roy Brereton, 94 Teignmouth Rd, Clevedon, Avon BS21 6DR.

Essex sub-group: Dave Walker, 22 Kempton's Mead, Potters Bar, Herts EM6 3HZ.

London sub-group: Jeremy Davis, 6 Elmcroft Crescent, Harrow, Middlesex HA2 6HN.

Northern Ireland: Billy Turkington, Fairyhill, Rostrevor, Newry, Co., Down BT34 3BB.

Qubbesoft PD library. Contact: Ron Dunnett, 38 Brunwin Rd, Rayne, Braintree, Essex CM7 5BU.

SCOTLAND

Scottish QL Users Group Contact: Alan Pemberton, 65 Lingerwood Rd., Newtongrange, Midlothian EH22 4QQ. Newsletter.

HOLLAND

Sin_QL_Air (Netherlands) Membership Secretary: Bob Visser, Snelrewaard 6, 2904 SN Capelle, a/d Ijssel, Netherlands. Magazine: *Quasar*. Editor: C H M Biemans, Elzenstraat 5, 5461 CL Veghel, Netherlands.

GERMANY

Sinclair QL User Club eV (Germany) Foreign Contact: Franz Herrmann, Talstrasse 21, d-W5460 Ochenfels, West Germany. Magazine: *Quasar*.

SPAIN

Qliper Editor: Marcos Cruz, Acacias 44, 28023 Madrid, Spain. Magazine: *Qliper*.

DIY TOOLKIT

Simon Goodwin shows how to change the names – and colours – of keywords, and then track them down.

This *DIY Toolkit* project extends QL SuperBasic in three ways. ALIAS, CODEVEC and INVERSE have been tested with *Minerva, Argos* and *Turbo* tasks. They should suit all Qdos systems and compilers.

The simplest new command is INVERSE, which swaps the foreground and background colours used to display text in a window. Many computer displays support 'inverse video', showing dark characters on a light background, rather than the reverse. It is a good way to highlight selections, whatever the colours in use.

The name comes from the ZX Spectrum print attribute INVERSE, but the QL implementation swaps INK and STRIP colours at every call; ZX BASIC INVERSE 1 and INVERSE 0 turned colour swapping on and off. The optional channel parameter defaults to #1.

You can do the same with INK and STRIP commands, but then you need to read the 'previous' colours with DIY functions like CHBASE and CHAN_B%. Unless you are careful you end up with both colours set to the same value, and invisible text!

INVERSE is simple, easily reversable, and preserves the original contrast. It is a great way to add emphasis in menu displays and prompts.

ALIAS lets you give new names to QL commands and resident functions, like this:

```
ALIAS "INVERSE" TO INVERT
ALIAS 'EXECUTE_A' TO TRY
ALIAS "SEARCH MEMORY" TO SCAN
ALIAS 'CODEVEC' TO VECTOR
ALIAS 'OPEN' TO OUVRE
```

Users unaccustomed to English can translate much of SuperBasic into French, German, Spanish or whatever; the only restriction is the 26 character Ascii alphabet common to all QLs, which prohibits the use of accents in resident names.

ALIASed names survive NEW and LOAD. They work at exactly the same speed as the names they mimic, since they use the same code. They are permanent, and consume no extra ram apart from eight bytes in the Name Table and the characters and length in the Name List. You can get rid of them with FORGET.

ALIAS forms a set with the resident function-creators SET and ALTER, the FORGET routine in DIY Toolkit Volume B, and *Task Commander,* the Resident Procedure generator presented last month. It is another way of expanding the QL Name Table to make the system easier to use and program. The Name Table is an efficient mechanism that can cope with up to 64K of names with no less of speed, once lines are tokenised.

ALIAS only works on resident procedures and function names, so it cannot provide substitutes for all the words in QL programs, but it can be convenient if you find yourself mistakenly typing commands for other computers while using the QL or an emulator. Why teach yourself, when you can teach the QL on your behalf? For instance, SAM and Spectrum users might use:

```
ALIAS "DELETE" TO ERASE
ALIAS 'DIR' TO CAT
```

Other substitutions will suggest themselves to those who use several computers. Some, like the Amiga CLI, match the QL commands quite closely already, though the QL is idiosyncratic in its use of names up to 36 characters long, replete with underscores!

The syntax of ALIAS may seem odd at first, but mimics COPY, RENAME and ALTER. The first name corresponds to an existing resident procedure or function. It appears in single or double quotes, while the second, new name is not quoted. Thus the code can alias any existing function name, without fetching its value instead, and the string to be matched may be an expression.

Each name may consist of up to 255 digits, underscores, small or capital letters (considered equivalent). The second parameter must be an unset name. The QL reports Bad Name unless the first name corresponds to a resident procedure or function.

Other values can move around, allocated and de-allocated on the variable values heap inside *SuperBasic;* it would be dangerous to ALIAS such names, as you might end up with an invalid pointer to space that had been re-used.

ALIAS works fine with the hundreds of names resident in every QL Rom, DIY Toolkit, *SuperToolkit,* and common extensions in disk systems. A few QL names are implemented as 'keywords', rather than resident procedures or functions; these include DEFINE, LOCAL, NEXT, EXIT, WHEN, and variants of ON, END and GO.

If you try to ALIAS these you get a Not found error because those names do not appear in the name table. The same goes for the separator TO, and operators like INST, OR and MOD. The only way to change these is to replace your rom.

The ALIAS command should not be used inside Supercharged or Turbo programs, as they contain no name list, but ALIAS works fine declaring names that are expected by compiled tasks. If a task reports 'name is redefined', type NEW to clear out the name list, and try again. NEW, LRUN and CLEAR do not discard ALIASed names.

ALIAS can be useful when running tasks compiled for some other system. A task expecting the Thor SYS_VARS function stops, reporting 'SYS_VARS is not defined' if run on a normal QL. It would be tricky or tedious to re-compile or patch the file to remove the reference. It is easier to load the DIY equivalent SYSBASE, and type:

```
ALIAS "SYSBASE" to SYS_VARS
```

Eureka! You can use the same principle to emulate missing commands with functional equivalents from other Toolkits.

The majority of names can be ALIASed in compiled programs without trouble, but you may miss out on updates or optimisations unless you are careful in your choices.

Turbo and Supercharge replace calls to some 'standard' resident procedures and functions with optimised routines from the compiler library. They identify the keywords to be optimised by name and position in the name table, so you may get odd results if you use unconventional names.

An alias of the PRINT command will work, but the rom routine is slower and displays only seven digits of precision, rather than the nine digits of the compiler library routine. Likewise MOVE_MEMORY, BLOCK, CODE, CHR$, LEN, PI, PEEK and POKE are optimised, so your task will go more slowly if you use ALIASed equivalents.

RUN, INPUT, READ, EOF, CLEAR and DIMN work differently in compiled tasks, so you should expect strange results if you call them by new names.

STOP and NEW will have no effect if you use aliased equivalents in a compiled program. These commands normally work by setting a 'stop code' at offset 140, BV.STOPN, among the SuperBasic Variables; the interpreter checks this word after interpreting each statement, but compilers don't do anything so silly; they

recognise the names, and substitute special code for STOP and NEW.

An aliased version of OPTION_ CMD$ will always return a null string. You need the original name to get the desired result. The same goes for other Turbo directives like REFERENCE, IMPLICIT%, PROCEDURE, FUNCTION and RETRY_ HERE.

Compilers replace calls to RESPR with library code to allocate Common Heap memory. An ALIAS of RESPR will not be replaced, so it will give a 'not complete' error if called from a task. Minerva roms fix this by using heap space automatically if the RESPR area cannot expand.

You may come across a few programs where alternative names like SINE and LOG will use the new code if you issue the ALIAS commands after loading. The new names will use the standard rom routines if you use ALIAS before any maths routines are loaded.

The function CODEVEC returns the address of the machine code routine that performs any resident SuperBasic procedure or function. You can use this address to disassemble the routine, or to set a breakpoint, allowing you to intercept calls to the keyword and step through the code.

Which is easier to type? CODEVEC ("ED") or BPEEK_ L (BPEEK_ L (24) + LOOKUP% ("ED") * 8 + 4)? This function could be replaced by a succession of calls to existing DIY Toolkit functions, but I find it useful and expect it to save typing for anyone exploring new resident code.

CODEVEC is short for CODE_VECTOR; you can of course ALIAS it to anything else you like. To save more typing, ALIAS 'CODEVEC' TO CV! I chose the name CODEVEC because it is easy to remember, yet unlikely to clash with existing variable names.

You can use CODEVEC to detect names that correspond through ALIAS, or ones that share code because of Toolkit updates:

```
DEFine FuNction CODEMATCH (a$, b$)
RETurn CODEVEC (a$) = CODEVEC (b$)
END DEFine
```

Listing two starts with the usual hex loader, the same each month, unless you write in to say you prefer the decimal line version used for *Task Commander*. The DATA corresponds to the assembly code in listing 1, but can be entered relatively quickly.

Check and correct the DATA lines if the loader reports 'Checksum incorrect' – otherwise, type the name of the file you want to create, eg FLP1_ALIAS_CODE. Before using ALIAS, INVERSE or CODEVEC you load the code from disk, like this:

```
X=RESPR (434)
LBYTES FLP1__ALIAS_CODE, X
CALL X
```

Alternatively you could use LRESPR, LINKUP to load and link the file, or ALLOCATION to reserve memory if tasks are running. Avoid ALCHP as the space it reserves does not remain allocated when you load a new SuperBasic program.

You do not need to retype the code from **Listing one** or **Listing two.** Binary code, original assembler source and documentation for ALIAS, INVERSE and CODEVEC are available now on 3.5 or 5.25 inch disks, or your microdrive cartridge.

Ask for DIY Toolkit volume **A for ALIAS**, the latest of 18 volumes of utilities and extensions, available from Richard Alexander at Cwm Gwen Hall, Pencader, Dyfed, Cymru SA39 9HA, or ring (0559) 384574. One volume costs £7, two cost £10 and so on, up to £58 for the full 1988-1991 set.

I have updated two volumes with *Task Commander* examples: the MAP command joins volume H, showing the exact allocation and ownership of memory on the heap, in task space, and elsewhere. Volume Q gains CHANS, a SuperBasic keyword that shows details of all the channels in use at any time.

Listing one is the program assembled into ALIAS_CODE. When you CALL the first instructions they point A1 at the table labelled DEFINE, read the BP.INIT vector word from ROM, and jump to that routine. It links the names and addresses at the end into SuperBasic.

The next routine deals with INVERSE. The default is INVERSE 1; brackets, hashes and other punctuation are optional. The most likely error reports are channel not open and bad parameter, if the channel is not a CON or SCR window.

The Qdos call SD.EXTOP is used to find the details of the window. INVERSE says bad parameter if you apply it to a MEM channel. The EXTOP works but the instructions to set the 'ink' and 'strip' fail.

If you are looking for a simple project, consider adapting INVERSE to set both the PAPER and STRIP colours. This takes two Qdos calls in machine code; the SuperBasic PAPER command sets both to the same value.

It is not enough just to swap the ink and paper values. EXTOP reads the colour values, avoiding *QRam, QPac* and Thor windowing variables, then calls SD.SETST and SD.SETIN to set the colours.

D1 is the only data register passed both ways by EXTOP. I use it to return both colours, exchanging the high and low 16 bit halves with a couple of SWAP instructions.

Two system calls update the colour masks SD.SMASK and SD.IMASK at offsets 48-65 in the standard channel defini-

```
* QL WORLD DIY TOOLKIT - ALIAS, CODEVEC and INVERSE
* Version 0.3, Copyright 1991 Simon N Goodwin.
*
start       lea.l   define,a1    Point A1 at definition
            move.w  $110,a2      BP.INIT vector
            jmp     (a2)         Link code to SuperBASIC
*
* INVERSE [ #ch% ] - exchanges channel INK & STRIP colours
*
inverse     moveq   #1,d0              Assume channel 1 at first
            cmp.l   a3,a5              Any parameters?
            beq.s   pick_chan
            move.w  $112,a2            Vector to get integers
            jsr     (a2)               CA.GTINT
            bne.s   give_up
            subq.w  #1,d3              One parameter expected
            bne.s   bad_param          Otherwise, complain
            move.w  0(a1,a6.l),d0      Get BASIC channel No.
            bmi.s   what_chan          It must be 0 or more
pick_chan   mulu    #40,d0             Scale for Channel table
            add.l   48(a6),d0          Add the base offset
            cmp.l   52(a6),d0          Check it is within table
            bge.s   what_chan          Reject if past the end
            move.l  0(a6,d0.l),d0      Pick up the channel ID
            bmi.s   what_chan          Negative if closed
got_id      move.l  d0,a0              Now A0 is channel ID
            lea.l   colour_ext,a2      A2 -> extension code
            moveq   #-1,d3             Infinite timeout
            moveq   #9,d0              SD.EXTOP trap key
            trap    #3                 Call QDOS IO system
            tst.l   d0                 Check for any error
            bne.s   give_up
            moveq   #40,d0             SD.SETST trap key
            trap    #3                 Set STRIP colour in D1.B
            swap    d1
            moveq   #41,d0             SD.SETIN trap key
            trap    #3                 Set INK colour
give_up     rts
*
colour_ext  moveq   #0,d1              Clear odd bytes of result
            move.b  69(a0),d1          Pick up STRIP colour byte
            swap    d1                 Prepare other half of D1
            move.b  70(a0),d1          Pick up INK colour byte
            moveq   #0,d0              OK, no error
            rts
*
* ALIAS "OLD_NAME" TO NEW_NAME  for Resident Procs & Functions
*
alias       lea.l   16(a3),a4
            cmpa.l  a4,a5              Two parameters please
            bne.s   bad_param
            tst.b   8(a3,a6.l)         Check type of parameter 1
            bne.s   bad_param          Reject unless it is unset
            moveq   #0,d5
            move.w  10(a3,a6.l),d5     Pick up the name's index
            bmi.s   bad_param
            lsl.l   #3,d5              Scale: 8 bytes per entry
            add.l   24(a6),d5          Add NT base offset
            move.l  d5,-(a7)
            lea.l   8(a3),a5           Isolate remaining parameter
            bsr.s   lookup             Evaluate parameter 2
            move.l  (a7)+,a5
            bne.s   bad_exit
            move.w  0(a2,a0.l),0(a6,a5.l)
            move.l  4(a2,a0.l),4(a6,a5.l)
            moveq   #0,d0
            rts
```

tion. These masks determine the pattern of bits set in video memory, and hence the colour displayed.

INK 0 is the simplest case. It clears the mask to zero. In MODE 4, INK 7 sets a mask of all bits set. This is the video memory pattern that corresponds to a white display.

The masks depend on the screen mode, as well as the colour and stipple. The first word of the mask is used on even numbered lines, the second on odd lines, to give stipples two pixels deep. In MODE 4, INK 7, 0, 1 stores the masks $FFFF and $0000 in SD.IMASK. In MODE 8, the same command stores $AAFF and 0, unless you select FLASH 1.

There are many cases to consider, and they might vary between machines; it is best to use normal Qdos colour codes, 0-255, and leave calculation of the masks to your system.

ALIAS works rather like ALTER from May 1991's *QL World*. It converts an unset name into a resident function – or procedure, in this case – by setting the first, third and fourth words in the corresponding name table entry. The second word is unchanged. It points to the name text in the name list.

Phil Spink observes that Unix programmers may prefer to type ALIAS NEW TO "OLD", in the style of an assignment. The DIY code is flexibly written so it can process the parameters in either order, with the minimum of tailoring. If the fourth and seventh lines of ALIAS use offsets 8 and 10 the unset parameter is the second one. Offsets 0 and 2 specify the first parameter.

The other parameter must be the 'old' name string. LEA.L 8 (A3),A5 singles out the first parameter for string LOOKUP; alternatively LEA.L 8 (A3),A5 selects the second. The DIY disk includes code and source to process parameters either way round.

CODEVEC calls LOOKUP to find its parameter, then plucks the relevant address from the name table with a single instruction. DIY code must work on all QLs and compatible systems, and most do not allow long integer function results, so it converts the value in D1 into a floating point value. The NORMALISE loop shifts the binary value of D5 to the 'normal' position at the left, adjusting the exponent in D4 accordingly.

CODEVEC must check that there are six bytes for its result. The slow call to BV.CHRIX could be avoided if the name is at least three characters long, as the result will fit in the space previously allocated to the string on the maths stack.

The last machine code subroutine, LOOKUP, resembles the eponymous keyword code. This version shows how two extensions can share code to fetch and analyse a parameter. It resets BV.RIP so that memory is not left allocated on the Maths Stack after the parameter has been processed.

```
_param  moveq   #-15,d0         Signal BAD PARAMETER error
_exit   rts                     Error code is in D0
t_chan  moveq   #-6,d0          CHANNEL NOT OPEN error
        rts

ODEVEC("NAME") returns code address for any Resident NAME

evec    bsr.s   lookup          Try to find the name
        bne.s   bad_exit
        move.l  4(a0,a2.l),d1   Pick up its code address

ETURN_FP stacks D1.L in SuperBasic floating-point form

urn_fp  move.w  d1,d4           D4 will be exponent
        move.l  d1,d5           D5 will be mantissa
        beq.s   normalised      Zero is a trivial case
        move.w  #2079,d4        First guess at exponent
        add.l   d1,d1           Already normalised?
        bvs.s   normalised
        subq.w  #1,d4           No, halve exponent weight
        move.l  d1,d5           Double mantissa to match
        moveq   #16,d0          Try a 16 bit shift
ealise  move.l  d5,d1           Take copy of mantissa
        asl.l   d0,d1           Shift mantissa D0 places
        bvs.s   too_far         Overflow; must shift less
        sub.w   d0,d4           Correct exponent for shift
        move.l  d1,d5           New mantissa is more normal
_far    asr.w   #1,d0           Halve shift distance
        bne.s   normalise       Try shift of 8, 4, 2 and 1

heck there's six bytes of space for the result

ealised moveq   #6,d1           No. of bytes needed
        move.w  $11A,a0         BV.CHRIX vector
        jsr     (a0)
        move.l  $58(a6),a1      Get a safe A1 value
        subq.l  #6,a1
        move.l  a1,$58(a6)      Grab 6 more bytes
        move.l  d5,2(a1,a6.l)   Stack mantissa
        move.w  d4,0(a1,a6.l)   Stack exponent
        moveq   #2,d4           Floating point result
        moveq   #0,d0
        rts

OOKUP "string parameter" -- NT entry @ (A2,A0) or error

kup     move.w  $116,a0         Fetch CA.BTSTR vector
        jsr     (a0)            Fetch string parameter
        bne.s   bad_exit        Quit unless it worked
        subq.w  #1,d3           Check for ONE parameter
        bne.s   bad_param       Moan, otherwise
        move.w  0(a1,a6.l),d0
        beq.s   bad_param       Reject a null parameter
        move.w  d0,d5           Save length for later
        lea.l   2(a1),a5        Save offset of text

quate UPPER/lower case; set bit 5 of parameter bytes

        moveq   #32,d7          Case conversion mask
        moveq   #1,d2           Odd/even length mask
        and.l   d0,d2           D2=1 if length is odd
k_case  or.b    d7,2(a1,a6.l)
        addq.l  #1,a1           Advance through text
        subq.w  #1,d0           Count down length
        bne.s   lock_case       Convert all characters
        lea.l   2(a1,d2.l),a1
        move.l  a1,$58(a6)      Tidy the RI stack
```

```
*
* Now find SuperBASIC task (0,0) and its Name Table
*
                moveq   #0,d2       Search entire task tree
                moveq   #0,d1       Look for SuperBASIC
                moveq   #2,d0       MT.JINF Trap key
                trap    #1          A0 := base of task 0,0
                move.l  a0,a2       A2 -> BASIC system vars
                move.l  24(a2),a0   A0 -> Name Table Start
                move.l  28(a2),d0   D0 -> Name Table End
                move.l  32(a2),d3   D3 -> Name List Start
*
* Scan the Name Table for names with the right length
*
next_name       move.w  2(a0,a2.l),a3  Pick up offset in NL
                adda.l  d3,a3       (A3,A2.L) -> Name
                cmp.b   0(a3,a2.l),d5  Compare length
                beq.s   got_length  Length matches!
advance_nl      addq.l  #8,a0       Advance through NL
                cmp.l   a0,d0       Stop at the end
                bhi.s   next_name
                moveq   #-7,d0      Signal NOT FOUND
                rts
*
* Check the name text to see if it matches the parameter
*
got_length      move.b  1(a3,a2.l),d4
                or.b    d7,d4       Ensure consistent case
                cmp.b   0(a5,a6.l),d4
                bne.s   advance_nl  Mismatch, try another
                move.w  d5,d6       Save residual length
                subq.w  #2,d6       DBRA count for the rest
                bmi.s   found_it    Found name, length 1
                move.l  a5,a4       D4 & A4 are temporary
check_name      move.b  2(a3,a2.l),d4
                or.b    d7,d4       Convert case of name
                addq.l  #1,a3       Step through Name List
                addq.l  #1,a4       Step through parameter
                cmp.b   0(a4,a6.l),d4
                dbne    d6,check_name
                bne.s   advance_nl  Name mismatch, go on
*
* Check that the name is a Resident Procedure or Function
*
found_it        move.b  0(a0,a2.l),d1
                subq.b  #8,d1       Expected type code 8 or 9
                bmi.s   wrong_type
                subq.b  #2,d1       Check type code <= 9 (!)
                bpl.s   wrong_type
                moveq   #0,d0       No error, CODEVEC is in D1
                rts
*
wrong_type      moveq   #-12,d0     BAD NAME error code
                rts
*
define          dc.w    2           Two procedures
                dc.w    alias-*
                dc.b    5,'ALIAS'
                dc.w    inverse-*
                dc.b    7,'INVERSE'
                dc.w    0
                dc.w    1           One function
                dc.w    codevec-*
                dc.b    7,'CODEVEC'
                dc.w    0
                end
```

```
100 REMark Sinclair QL World HEX LOADER v 3
120 :
150 CLS: RESTORE : READ space: start=RESPR(space)
160 PRINT "Loading Hex..." : HEX_LOAD start
170 INPUT "Save to file...";f$
180 SBYTES f$,start,byte : STOP
190 :
200 DEFine FuNction DECIMAL(x)
210 RETurn CODE(h$(x))-48-7*(h$(x)>"9")
220 END DEFine DECIMAL
230 :
240 DEFine PROCedure HEX_LOAD(start)
290 byte = 0 : checksum = 0
300 REPeat load_hex_digits
310    READ h$
320    IF h$="*" : EXIT load_hex_digits
330    IF LEN(h$) MOD 2
340       PRINT"Odd number of hex digits in: ";h$
350       STOP
360    END IF
370    FOR b = 1 TO LEN(h$) STEP 2
380       hb = DECIMAL(b) : lb = DECIMAL(b+1)
390       IF hb<0 OR hb>15 OR lb<0 OR lb>15
400          PRINT"Illegal hex digit in: ";h$ : STOP
420       END IF
430       POKE start+byte,16*hb+lb
440       checksum = checksum + 16*hb + lb
450       byte = byte + 1
460    END FOR b
470 END REPeat load_hex_digits
480 READ check
490 IF check <> checksum
500    PRINT"Checksum incorrect. Recheck data.":STOP
520 END IF
530 PRINT"Checksum correct, data entered at: ";start
560 END DEFine HEX_LOAD
570 :
580 REMark Space requirements for the machine code
590 DATA 432
600 :
610 REMark Machine code data
620 DATA "43FA018A34790000","01104ED27001BBCB"
630 DATA "67143479000001l2","4E92663853436 67E"
640 DATA "3031E8006B7CC0FC","0028D0AE0030B0AE"
650 DATA "00346C6E20360800","6B68204045FA0018"
660 DATA "76FF70094E434A80","660A70284E434841"
670 DATA "70294E434E757200","12280045484112 28"
680 DATA "00467004E7549EB","0010BBCC66304A33"
690 DATA "E808662A7A003A33","E80A6B22E78DDAAE"
700 DATA "00182F054BEB0006","61682A5F66123DB2"
710 DATA "8800D8002DB28804","D8047004E7570F1"
720 DATA "4E7570FA4E75614A","66F62230A8043801"
730 DATA "2A01671C383C081F","D281691453442A01"
740 DATA "70102205E1A16904","98402A01E24066F2"
750 DATA "7206307900000011A","4E90226E00585D89"
760 DATA "2D4900582385E802","3384E80078027000"
770 DATA "4E7530790000116","4E9066A45343669E"
780 DATA "3031E80067983A00","4BE900027E207401"
790 DATA "C4808F31E8025289","534066F643F12802"
800 DATA "2D4900587400 7200","70024E412448206A"
810 DATA "0018202A001C262A","00203670A802D7C3"
820 DATA "BA33A800670A5088","B08862EE70F94E75"
830 DATA "1833A8018807B835","E80066EA3C055546"
840 DATA "6B16284D1833A802","8807528B528CB834"
850 DATA "E80056CEFFF066CE","1230A8005101 6B08"
860 DATA "55016A0470004E75","70F44E750002FED8"
870 DATA "05414C494153FE76","07494E5645525345"
880 DATA "00000001FF020743","4F44455645430000"
890 DATA "*",36114
```

LOOKUP reads its parameter as a SuperBasic string, to avoid problems with function names. The check on the value of D3, the number of string parameters, is only relevant when called from CODEVEC; ALIAS sets A3 and A5 to delimit a single parameter.

As written, LOOKUP searches for names in the tables of SuperBasic task (0,0). This suits Turbo and Supercharge, but users of *QLiberator* and *Minerva* 1.8x, which create extra name tables, might want the scan to examine the current task.

To achieve this, remove the five lines that point register A2 at task (0,0) and replace subsequent references to A2 with A6, the current task base. This entails changes to eleven operands, and one comment. The resultant code is ten bytes shorter. The DIY disk includes both implementations.

PSION SOLUTIONS

Why does Quill V2.3 take the law into its own hands when you ask it to save a new file under an existing file name? Why won't the Psion suite save one reader's files to disk? Howard Clase catalogues some Quill bugs, looks at Ron Massey's alterations to install_bas, and offers a modification to his own program FTidy.

I've just come a cropper with Quill (v 2.3). I realise it wouldn't have happened had I backed up files but even so it looks like a serious bug to me.

If one tries to save a _doc file with an existing filename one is asked whether the old file is to be deleted. Pressing <ESC> returns one to the original prompt, for example 'Save,flp2_letter_doc'. If one then attempts to save the file by overwriting this prompt by typing in, say, 'letter2' Quill takes no notice but deletes the original file and saves the new one with the old name.

Hilary Snaden
Portishead
Bristol

I can confirm that this happens with both versions I have available too (QLWP 2.1 and *Quill* 2.35). Hilary Snaden's careful observation explains the loss of more than one of my files! There seems to be no way to cure it without modifying the code for Quill itself, so it just has to be lived with. The only answer if you change your mind while saving is to remember to press <ESC> TWICE and then repeat the entire save sequence with the new name.

Bugs

This is probably an opportune introduction to a catalogue of other bugs and inconveniences in Quill. There is another serious bug in all versions: if you use the Erase or Copy functions the program partially loses track of cursor movements. Pressing <SHIFT down-arrow>, which usually moves the cursor to the start of the next paragraph, has an erratic effect. More seriously, repeated use of <up-arrow> may lock up Quill altogether without warning. In my experience it is only <up-arrow> that causes the problem and it is only Quill that is locked, not the whole QL, so if you are multitasking you can still get at your other programs, ramdisk files, etc.

Pointers

Contrary to some advice I have read, you cannot reset the pointers by going to the top or bottom of the _doc. You can avoid using up-arrows by using <SHIFT up-arrow> to get to the top of the paragraph and coming down line by line, but it's probably safer to save and reload your document after using either of these two commands as I always do. If you can use *Toolkit II* to generate 'macros', this only involves a couple of keystrokes. Note: It's the reloading that resets the pointers; you cannot get away with saving alone if you are going to do any more work on the file.

Printer

Another way to lock up Quill, and your printer port as well, is to ask Quill to print something without having a printer ready and waiting at the other end of the printer cable. The <ESC> key has no effect in these circumstances. If it is just a case of pressing the on-line button on the printer, then the situation is recoverable, but switching on a printer or plugging in a printer cable while the QL is running, risks a chip-destroying surge. That said, I sometimes do switch on my printer after the QL, but I have taken care that it is plugged in to a different main circuit. That hasn't prevented my wife from starting up the Electrolux on the same circuit as the QL, but without causing any problems – so far!

There are a few other quirks of Quill that beginners may come across, which, while they aren't absolute disasters, can be frustrating if you don't know what is going on.

Headers

Firstly headers and footers. The default settings when Quill is switched on are: no header, and a footer that prints 'page 1', etc. at the bottom centre of each page. Unfortunately the default page design expects the top of the sheet to be at the print head, not at the tear off bail, which is where it is convenient to put it. This means that if you are using form feed paper, 'page 1' appears not at the bottom of page one, but at the top of page two. You can get around this by putting the perforation at the print head – but this means wasting one sheet every time you print a letter.

My personal solution to this is to get into 'Design' <F3,D> and set the upper margin to 0 and the bottom margin to 9. This is about right for the top of the sheet at the bail. I also prefer to have my page numbers at the top rather than at the bottom, so I set the footer to "none" and the header to "nnn", which prints the page number in ordinary Arabic numerals.

Footer and header work similarly: after you have decided whether you want your message left, right or centre you are presented down at the bottom left corner of the screen with the present message if there is one, otherwise the line is blank waiting for you to type it in. The enhanced Psion line editor I mentioned in an earlier article is available, so you can use SHIFT with the arrow keys, etc. If you include "nnn" in your message it is replaced by the page number in Arabic numbers. ('rrr' gives Roman numerals and 'aaa' alphabetical labelling – upper case, eg 'NNN', has the same effect. You have to type them in, not say them.)

Bold

The last question asked when you are resetting footers or headers is 'bold'. While Quill is able to remember the position and content of your message it seems to suffer from amnesia when it gets to the typeface – so you have to press 'n' every time

if you don't want your header in the default **bold** type.

This leads me to another problem which I met when I had a fading ribbon and tried to compensate by printing a whole article in bold type. Quill thinks 'normal' is whatever the main text is in, and doesn't send any typeface codes to the printer when printing header and footers, but I had set the header to bold. The result was that, having printed the header in bold, it sent the code to turn bold off, and the main page came out in normal type. When I set the header to 'normal' it didn't change the typeface and everything came out as I wanted it – all bold! You've just got to look at things the way Quill sees them.

Def_tmp files

I'm writing this while on a short sabbatical in England using a basic 128K QL and a tv set. Perhaps all contributors should do this from time to time, since I've discovered another annoyance that I never would have met on my 640K disk drive QL at home (NB Ms Editor, that's Newfoundland, not Nova Scotia). *Let's just say Canada*. The code for Quill itself takes up so much memory in the QL that there is only enough room for about 700 words left on an unexpanded QL, so when this is full Quill sets up a special file, def_tmp, on mdv2_. It keeps the whole of your text in this and only keeps the bit you seem to be currently interested in in the active memory (in computerese, it uses the mdv tape as virtual memory).

Merging

This can slow things down, while the mvds whirr as you move about your document, so I am trying to write my article in short bits, only merging them together when everything is finished. Obviously, since the tape in mdv2_ contains part of your doc, you cannot take it out to save a second copy as a backup, but you can take out the Quill tape from mdv1_ and make a copy on that microdrive. (Remember to put the Quill tape back before you print.) When you leave Quill with quit, or zap a file, it should delete the def_tmp file. However, if you load a new shorter file without 'zapping' the original doc then the def_tmp file remains on your cartridge taking up unnecessary space. You can delete it from within Quill using the 'files' command <F3,F3,F,D>; for once Quill seems to be intelligent enough to allow this only if the def_tmp file is not part of your current doc.

If you do forget, and try to change cartridges in mdv2_ while there is an active def_tmp, Quill doesn't give you a gentle reminder, it says: "Fatal error – restarting Quill", and your document is lost. Even though it seems to be starting again from scratch it gets very possessive about the original cartridge and won't accept any other – it's best to reset at this point.

Configured or confused?

Mr Raymond Fowles of Liverpool has upgraded his QL to disk, but is having problems getting his Psion suite to run on his new system. In particular he has been unable to save any of his files to disk.

Mr Fowles' problem arises because each of the Psion programs has a number of 'default' devices built into its software. If you are still using the original versions, you will probably have noticed that when loading or saving a file in any of the Psion suite you do not have to tell it which microdrive to use as you do in Basic. It just assumes you mean mdv2_; although you can over-ride this default by typing the device name before the filename, eg if you type in "mdv1_letter" it will save it to mdv1_. (I wouldn't recommend this as a regular habit, as there isn't very much extra room on the Quill cartridge.)

Whirr

Also if you ever need help or want to print a document you will notice mdv1_ whirr for a bit before anything happens. It is because of the built-in defaults in the code that Quill knows where to look for the appropriate files.

If you put your Quill cartridge into mdv1_ *after* pressing F1/F2 and type "copy mdv1_quill to scr_" at the keyboard you will see the code for Quill spew out onto your screen; most will be gobbledegook, but occasionally recognisable words will appear, and amongst them you will see mdv1_, mdv2_, mdv1_ in close succession. If you have an (or even THE) editor or a monitor you will be able to find them more easily and even alter them, for this is where the names of the defaults are hidden.

Config_bas

There is no need to spend money on an editor though; there is a program called config_bas supplied with the Psion suite. In fact there are no less than three identical copies of it, one on each master, apart from Archive. You can use any copy to configure any member of the suite. As far as I can see there is no significant difference in the various versions of config_bas that come with the different versions of the Psion suite, but, if, like me you have more than one version, it is probably safest to use the right one – I haven't tested them all exhaustively.

Reconfigure

To re-configure your copies of the Psion suite, first reset your QL and press F1/F2 before inserting your Psion cartridge into mdv1_. Then enter the command "LRUN mdv1_config_bas". In operation config_bas is easier to use than install_bas; just follow instructions. First it asks you for the name of the program you want to configure and where it can be found (mdv1 and mdv2 are the only options, so you will have to copy your reconfigured program to disk later). You can reset the default devices for all four Psion programs and for Abacus and Archive there is the additional option of changing the sort order – I'll come to that later.

The three device defaults in order are:

1. The device that contains 'system information'; in practice this means the location of the printer drivers – printer_dat or gprint_prt in the case of Easel.

2. Where you keep data files: eg _doc files in Quill or _aba files in Abacus.

3. The location of the _hob files which are called up when you press <F1> for HELP!

Dual Disk

If you have a dual disk system then the obvious thing is to simply swap flp1_ for mdv1_ and flp2_ for mdv2_. If like me, you have only one disk drive then you can either set all three devices as flp1_, or use a ramdisk as I do. I set the 'system' device as ram8_, and have altered my boot to copy the appropriate _dat file to ram8_printer_dat before EXEC_Wing the Psion program. I have differently named _dat files for different purposes and use the backup facility, <F3, F3, F, B>, if I want to change ram8_printer_dat. N.B. it must always be called printer_dat in ram8_ whatever it's called on the disk – you can change the name when you copy or back it up.

If you run config_bas with Abacus or Archive you can also change the sort order, ie the alphabetical order used when strings are sorted. If you are using English only then you probably won't want to change it. This facility is primarily made available for users of other languages which use other sequences – for example, in German a-umlaut is treated as equivalent to a, but in Finnish it comes almost at the end after z, but before o-umlaut, and w and v are considered equivalent. (This might come in useful if you are ever using the Helsinki phone directory!)

There is an alternative way of dealing with running the Psion suite on disks – or any other programs that have built in references to microdrives – and that is to use the FLP-USE command which is present in almost all disk interface software. Edit your boot to include

the following line before any reference to mdv is made:

nnn FLP_USE MDV (where nnn is an appropriate line number.)

This statement makes 'flp' an alias for 'mdv' so an instruction to look for a file on mdv2_ is redirected to flp2_ for example. I prefer to call a disk a disk and not to indulge in euphemisms of this kind, as you can get into a muddle. Obviously you can only do this without using config_bas if you have dual disk drives, and you will cut yourself off from access to your microdrives themselves.

One More

Mr Fowles also refers to some alterations to install_bas proposed by Ron Massey in *QL World*. It is not necessary to do anything to install_bas when you convert to disks, just copy over the printer_dat files and they will work as they are. Install_bas enables you to design your own printer_dat files which the program refers to when it needs to know the special codes your printer uses to create effects like superscript, bold, etc.

Alterations

Ron Massey's alterations represent one approach to solving the problem of wanting to pick and choose between different versions of printer_dat for different situations – especially when you have more than one of the Psion Suite on the same disk. I go along with him as far as giving each version a distinctive name, but you can do that with the COPY command eg COPY mdv1_printer_dat to flp1_Quillnlq_dat.

His approach also involves altering the code of your Psion programs, and you cannot do that without an editor or some skill at writing SuperBasic. Even if you do follow his suggestions your new version is still restricted to one named _dat file. Using the backup facility in the way I describe above is more flexible and doesn't require any alterations except a bit of renaming of printer_dat files.

The main trouble with both config_bas and install_bas is that both are designed to operate with microdrives alone so you have to modify or create your files on microdrives and transfer them to disk, or use the FLP_USE command. It only requires a small amount of editing to make disk versions of these programs. I will investigate and report in a future article.

FTidy bug

Some readers have been having problems with a file handling program of mine called *FTidy* which was available through QL World's mdv exchange. This was primarily intended for QLers with disk drives and requires a small alteration to run on a QL with microdrives only. In order to manipulate the directory it has to be read into a temporary file, and the program as supplied tries to put this onto flp1_. To use FTidy with microdrives load the program FTidy_bas and type EDIT 640, you should get a line starting:

640 DATA "FLP1_": REMark...

Edit this to read:

640 DATA "MDV1_": REMark...

and press enter. Delete FTidy_bas from your working cartridge and save this modified version as FTidy_bas in its place. You should have no further problems, as long as there is a tape in mdv1_ with enough space for a short file – two sectors is probably enough. If you do have a ramdisk then use this rather than mdv1_, it is much quicker. If you still have problems then please write to me c/o Sinclair QL World.

This only applies to the original SuperBasic version of FTidy. A Q Liberated version, with a superbasic program to enable you to configure the devices has been sent in to the new microdrive exchange, but administrative problems have prevented it from appearing yet.

ARCHIVE

A=N+S-W=E+R-S

Archive's numerical abilities are really quite impressive. Numbers are stored to 14 significant floating point figures, to a maximum value of some 1.7E38 – by my reckoning that is plus or minus 170 billion billion billion!

The downside of this powerful feat is that Archive numbers still occupy enough space for such numbers (8 bytes), even if all you want to store are numbers between 1 and 10, or even just to store a zero. That does not matter on a small database, but in applications where large amounts of small numbers are involved, it can represent a huge waste of both *computer memory and disk/microdrive* space.

Another problem with Archive numbers is the inability to define arrays or sets of any sort. Each number has to be held in a variable of its own. This presents all sorts of problems, both in terms of the awkwardness of this kind of data handling, and in the inefficiency of space it can lead to. When devising a new record structure, you have to allocate every number variable you may need – even if in most of the records, it will be carefully storing zero to fourteen significant floating point figures.

This *Archive Answers* will not be a finished application. Instead it offers the Archive programmer several alternative sets of tools to help overcome this situation. They represent several different data structures, each of which may improve both the data access and data storage for particular kinds of problems. The data type used will be integers (whole numbers) in a range of plus or minus 36000 or so. This can be stored in two bytes of a normal text string. The two short procedures in **listing one** convert Archive numbers into these string integers, and back again. As with all *Archive Answers*, input into a procedure is via calling parameters, and output is via global variables: ANSWER for string, and ANSWER for numeric output.

Once the numbers are held as string values they can be concatenated together, so that one string variable can be used to hold up to 127 such values. Once in such a string variable, these numbers are very easy to deal with *en bloc*. It is easy to duplicate the whole set, by simply assigning it to another variable. You can also find the total number of elements, by simply dividing its length by two. And because Archive uses variable length string variables, they only occupy the space they need. What is not so easy is dealing with

Robin Stevenson offers the Archive user several alternative tools to assist number-handling.

Table One. Which procedures are needed by each data structure?

proc *struc*	Encode	Decode	Array *Dim*	Array Store	Array Get	Push	Pop	Pull	Array Insert	Array Cut
Fixed Length Array	Y	Y	Y	Y	Y					
Variable Length Array	Y	Y		Y	Y				Y	Y
Stack	Y	Y				Y	Y			
Queue	Y	Y				Y		Y		

Listing One

```
proc Encode;VALUE
  let VALUE=VALUE+32639
  let ANSWER$=chr(VALUE/256)+chr(VALUE-(int(VALUE/256)*256))
  endproc

proc Decode;ITEM$
  let ANSWER=(code(ITEM$(1))*256)+code(ITEM$(2))-32639
  endproc
```

Listing Two

```
proc ArrayDim;ELEMENTS
  let ANSWER$=rept(chr(127),ELEMENTS*2)
  endproc

proc ArrayStore;ARRAY$,ELEMENT,VALUE
  local LENGTH: let LENGTH=len(ARRAY$)/2
  if ELEMENT<=LENGTH and ELEMENT
    Encode;VALUE
    if LENGTH=1: return : endif
    if ELEMENT=1: let ANSWER$=ANSWER$+ARRAY$(3 to )
      else
      let ANSWER$=ARRAY$( to (ELEMENT*2)-2)+ANSWER$
      if ELEMENT<LENGTH
        let ANSWER$=ANSWER$+ARRAY$((ELEMENT*2)+1 to )
        endif : endif
    else : print "ArrayStore: element out of range"
    let ANSWER$=ARRAY$: endif
  endproc

proc ArrayGet;ARRAY$,ELEMENT
  if ELEMENT*2<=len(ARRAY$) and ELEMENT
    Decode;ARRAY$((ELEMENT*2)-1 to )
    else : print "ArrayGet: element out of range"
    let ANSWER=0: endif
  endproc

proc Show;ARRAY$
  local LENGTH,COUNT: let LENGTH =len(ARRAY$)/2
  let COUNT=1: while COUNT<=LENGTH
    ArrayGet;ARRAY$,COUNT
    print COUNT;":";ANSWER
    let COUNT=COUNT+1: endwhile
  print "An array of ";LENGTH;" elements."
  endproc

proc ArrayTest
  local COUNT:ArrayDim;10: let A$=ANSWER$
  print "Enter numbers between 1 and 10. 0 to finish"
  let COUNT=1: while COUNT<>0
    input ": ";COUNT
    if COUNT
      ArrayGet;A$,COUNT
      ArrayStore;ANSWER$,COUNT,ANSWER+1
      let A$=ANSWER$: endif
    endwhile
  let COUNT=1: print "Frequency:-"
  Show;A$
  endproc
```

the individual elements. The procedures developed here help overcome this in various ways. With the appropriate data-handling procedures, the strings can be treated as a variety of different data structures.

The simplest is probably the fixed length array, which always contains a fixed number of element values. More space efficient options include a stack – in which numbers can be added or removed from the top of a list – and a queue, adding to one and removing it from the other. The most flexible, but also the trickiest to keep under control is the variable length array. Data can be read, inserted or deleted from any point, and there is no need to waste storage space.

Each of these data structures has its benefits and limitations. It is important to match an application with the appropriate date structure, to maximize the benefits, and minimise the shortcomings.

The savings to be gained could be considerable. If you need to store 50 numbers per record, in 200 records, this would occupy some 80K by conventional means, and only 20K in fixed length arrays. If you needed up to 50 numbers per record, but on average only used 10 of them, it would still take 80K normally, while variable length arrays would use not much more than 4K.

Using structures

We shall now look at the nitty gritty of using these data structures. *Encode* and *Decode* do the maths of converting numbers to two-character strings and back, and are needed for all of the data structures. However you may never need to call these directly. Each data structure uses its own set of storing and extraction procedures, which obey a set of rules specific to the particular structure. While such rules can limit what any one array can do, they make the programming very much simpler. When designing the system, choose a data structure suited to the task and stick to it. In this way, you can actually gain more freedom in the use of the data, since you can be sure of what is going on. It is one of the paradoxes of programming, as of life, that as options (and so complexity) increase, so the effective freedom of choice tends to decrease.

The first data structure we shall look at is the fixed-length array. There are three procedures needed here: *ArrayDim* to dimension the array; *ArrayStore* to store numbers; and *ArrayGet* to retrieve numbers. The basic rule for a fixed-length array is (surprise, surprise) that you can't change the number of elements. When you dimension the array, you fix its length, and each element is assigned the value zero. This means it is quite legal to retrieve elements that have not had a number stored in them. The first three procedures of **listing two** perform these three tasks. The remaining two are demonstration procedures. *Show* will display the values held in any array passed to it. *ArrayTest* is a simple demonstration of one use of a fixed-length array. In this instance, there are ten elements in the array, and you are requested to enter as many values between 1 and 10 as you like. For each entry, the existing value for that element is read, added to, and restored. In this way it monitors the frequency of each number pressed.

In this example it is important that all the elements start off initialised at zero, so they can be added to, even on the first call. It is also useful to be able to deal with each element by number. The procedure doesn't

need to know which number was pressed. It can simply pass it on when getting and storing the frequency. One important point to note about these array strings is that they are not automatically put into your array variable. ArrayTest stores its array in A$. If you look at the occasions this is used, you will see that while you can pass A$ to a procedure (as in ArrayGet;A$, COUNT), it cannot pass it back. Instead the revised array is now in ANSWER$, which must be re-assigned to A$ (let A$=ANSWER$). This must be done after any procedure which alters the array. In this case that means ArrayStore and ArrayDim. It is an unfortunate weakness with Archive parameters. Also note that if A$ was a file variable, instead of a memory variable, you would also need to use the UPDATE command to make the change to A$ permanent.

There are many occasions when fixed-length arrays could be useful. Anything where a fixed amount of data arrives in a variable order is ideal. For example, your students each have six essays to do during the term. You could set up a database with a six-element array for each student record, each essay title corresponding to a number. As the essays are marked you could store the values against the relevant essay number. It would be fairly easy to devise procedures to search for unmarked essays (hopefully no student will get 0 percent as an actual mark), and to add up the totals for each student.

The stack

The next structure we shall look at is the stack. This is used a lot in low-level programming, and also in Forth, where it is the main data feature. The idea behind it is that elements can only be added to the top of the stack, or taken off the top (like a stack of plates in a cafeteria as the tutorials always say). Archive users are unlikely to want a multipurpose stack of the kind used in Forth, so rules to control stack use by different parts of a program probably don't apply. By limiting access to nothing but adding and removing, the programming is very simple, and it is very easy to keep track of the data.

The stack is accessed entirely by two procedures, *Push* and *Pop* – the traditional names given for adding and removing values. A stack is one of the simplest but also most limiting of data structures.

Similar in both form and use is a queue. The difference is that values are extracted from the bottom, instead of the top of the list. It is a first-in-first-out system. The procedures for a queue are *Push* (the same as for the stack) and *Pull* which removes the bottom value. One use for a queue might be to co-ordinate a real queue in a doctor's waiting room. As patients arrive, their IDs are added to one end of the queue. Then each ID can be taken from the front of the queue, to show who is next, and then automatically posted in the consultation record.

Either of these structures would also be ideal for any occasion where an unknown number of values need to be gathered, before being used to produce statistics. For example, an athletics event may wish to record lap times for each runner, in each race. Each runner's record would

Listing Three

```
proc Push;ARRAY$,VALUE
  if len(ARRAY$)<253
    Encode;VALUE
    let ANSWER$=ARRAY$+ANSWER$
    else : print "Push: stack full"
    let ANSWER$=ARRAY$: endif
  endproc
```

```
proc Pop;ARRAY$
  local LENGTH: let LENGTH=len(ARRAY$)
  if LENGTH>=2
    if LENGTH=2:Decode;ARRAY$: let ANSWER$=""
      else :Decode;ARRAY$(LENGTH-1 to )
      let ANSWER$=ARRAY$( to LENGTH-2)
      endif
    else : print "Pop: stack empty"
    let ANSWER$="": endif
  endproc
```

```
proc Pull;ARRAY$
  if len(ARRAY$)>=2
    Decode;ARRAY$
    if len(ARRAY$)=2: let ANSWER$=""
      else : let ANSWER$=ARRAY$(3 to ): endif
    else : print "Pull: queue empty"
    let ANSWER$="": endif
  endproc
```

```
proc QTest
  local COUNT
  print "Enter numbers +/-32600, ending with 0"
  let FIFO$="": let COUNT=1: while COUNT<>0
    input ": ";COUNT
    if COUNT
      Push;FIFO$,COUNT: let FIFO$=ANSWER$
      endif
    endwhile
  let LIFO$=FIFO$
  print "The numbers were:-"
  print tab 5;"FIFO"; tab 20;"LIFO"
  while len(FIFO$)
    Pull;FIFO$: let FIFO$=ANSWER$
    print tab 5;ANSWER;
    Pop;LIFO$: let LIFO$=ANSWER$
    print tab 20;ANSWER
    endwhile
  endproc
```

Listing Four

```
proc ArrayInsert;ARRAY$,ELEMENT,VALUE
  local LENGTH: let LENGTH=len(ARRAY$)/2
  if ELEMENT<=LENGTH+1 and ELEMENT
    Encode:VALUE
    if ELEMENT>1
      let ANSWER$=ARRAY$( to (ELEMENT*2)-2)+ANSWER$
      endif
    if ELEMENT<=LENGTH
      let ANSWER$=ANSWER$+ARRAY$((ELEMENT*2)-1 to )
      endif
    else : print "ArrayInsert: out of range"
    let ANSWER$=ARRAY$: endif
  endproc
```

```
proc ArrayCut ;ARRAY$,ELEMENT
  local LENGTH: let LENGTH=len(ARRAY$)/2
  if ELEMENT<=LENGTH and ELEMENT
    Decode;ARRAY$((ELEMENT*2)-1 to )
    if ELEMENT>1
      let ANSWER$=ARRAY$( to (ELEMENT*2)-2)
      else : let ANSWER$="": endif
    if LENGTH>ELEMENT
      let ANSWER$=ANSWER$+ARRAY$((ELEMENT*2)+1 to )
      endif
    else : print "ArrayCut: out of range"
    let ANSWER$=ARRAY$: endif
  endproc
```

```
proc Prune;ARRAY$
  local T,COUNT
  let COUNT-2:ArrayGet;ARRAY$,1: let T=ANSWER
  let ANSWER$=ARRAY$
  while COUNT<=len(ANSWER$)/2
    ArrayGet;ANSWER$.COUNT
    if ANSWER=T:ArrayCut;ANSWER$,COUNT
      else : let COUNT=COUNT+1: let T=ANSWER
      endif
    endwhile
  endproc
```

```
proc InsertTest
  local N,COUNT: let A$=""
  print "Enter numbers, +/-32000. End with 0"
  let N-1: while N<>0
    input ": ";N
    if N
      let COUNT=1: let ANSWER=-40000
      while COUNT< -len(A$)/2 and N>ANSWER
        ArrayGet;A$,COUNT
        if N>ANSWER: let COUNT=COUNT+1: endif
        endwhile
      ArrayInsert:A$,COUNT,N: let A$=ANSWER$
      endif
    endwhile
  Show;A$
  Prune;A$: let A$ -ANSWER$
  Show;A$
  endproc
```

include a stack, into which was added each lap time. At the end of the race, it would be possible to find the fastest, and average laps of each runner, and use the same system for races of different lengths.

Listing three contains the stack and queue procedures, plus a demonstration. This shows how the same data, once collected, can be used as either a stack or a queue, as needs require.

The final data structure offered here is the variable-length array. This uses both ArrayStore and ArrayGet from **Listing two,** adding procedures to insert and delete values anywhere in the list. It has no inherent rules in the way the others do, and is clearly of less use when the absolute position of the elements is important. However it is ideal where the relative value is significant. For example, you could insert each new number in ascending rank order, as demonstrated by the *InsertTest* procedure in **Listing four.** Once numbers are in order it is very easy to check for duplicated values. The *Prune* procedure does just this, and prunes out any duplicates it finds.

One important, but complex use for an array could be in a one-to-many relational database. Records in the main data file could store an array of reference numbers, for each of the subsidiary file records that relate to it.

When entering the listings initially, you should add all four together into one program, as there are overlaps in the procedures they require. However when you come to use them in your applications, you could remove those you do not require, for the particular structures you need. **Table one** shows which procedures relate to each of the data structures described.

A word of warning may be in order regarding printing these array strings on the screen. Because they may include values below CHR (32), they can interfere with the screen driver and write rubbish, or generate error messages. This is true even with a file variable using DISPLAY. You may need to customise the input screen, to remove such variables.

It would be fairly easy, by just altering the Encode and Decode procedures, to convert all these procedures to handle a different set of integers. If zero was positioned at true zero, instead of at 32639, you could store numbers from 0 to 65000 instead. If only very small numbers were needed, you could, with rather more wide-ranging changes, use a single byte for each number, to store integers from 0 to 255, for example. This would halve the storage area used.

There is also scope for using other data structures, or access rules from those described here. Archive programmers are probably not used to thinking in terms of bytes of data, or required integer ranges. The vastness of the built-in numeric variables shields us from that. However if your applications are too inefficient using these variables, the integer arrays described here could be the answer you need.

ArchEd Part II

Stephen Mitchell continues with the second part of his Archive editor.

In the last article we produced some file handling procedures that will enable you to link your dbf file to the *ArchEd* editor procedures that appear this month. This Editor runs within Archive and enables you to edit the fields of your file to their full 255 character length. ArchEd will handle a file containing any number of fields without the need to create special sedit screens. Arch Ed can also be incorporated within your existing Archive procedures. Nearly all this month's procedures are prefixed by the letters w2. Type in the procedures given in the listing using the Archive edit command and save it with the name given in the initial REMarks. Make sure you keep a security copy.

The task of the w2 procedures is to display each field in turn from the record selected by the file handling procedures and to enable individual fields to be edited as required.

Before proceeding it is necessary for the w2g1 procedure to be edited to the names of the fields in your file. A non-elementary file structure is given as an example in procedure w2g2. The parameter 'n' to this procedure simply indicates which field is to be assigned to the string variable s$. The fields of a file are numbered from zero, so if the fifth field of your file is called 'descriptions$' its field number is 4 and so is the statement:

if n=4: let description$=s$: endif

must be included in w2g1. A statement of this form must appear for each of the textual fields in your file with the field numbers set as appropriate. The example shows how to code this structure in a reasonably efficient manner. Note that a gap will be present for any numeric field defined in your file. The w2g2 procedure can be deleted once w2g1 has been amended to reflect your file.

As mentioned last month, if you have an expanding QL then you might like to try merging the editor and file handling procedures and running them as a single loadable program. ArchEd runs somewhat more efficiently in this form.

ArchEd is run in the normal way by typing 'run ArchEd ENTER'. The initial questions asked are those from the file handling procedures with which you should by now be quite familiar. When you have selected a record from your file you will be asked 'Do you want to EDIT this record?'. If the answer is 'yes' ArchEd will display the first textual field of the record. This field becomes the 'current' field and appears double-lined spaced at the top of the screen. The individual lines into which the current field is broken are known as 'components' and it is these that are edited by the sub-edit commands (see below). Of course, the field may be shorter than the component length in which case only one line is displayed. The current field is followed by the next field (if this is textual) single-line spaced. The following editing instructions are displayed at the bottom of the screen:

ENTER: Edit field (i/d/r/di=insert/ delete/replace/del & ins); (SHIFT) 1,2, ..., 10: Accept field edit ord skip (-)n fields (cyclic); F4/F5: Abandon/Commit all field edits to current record and Exit; Rp Space enter: re-edit current field when at next component; <3 SPACES>: Accept field edit when at next component.

Again, these are mostly self-explanatory but the sub-editing commands require further amplification by way of some examples.

To perform any editing on the current field you must press ENTER. This causes the cursor to appear under the first character of the first component of the current field. This left-hand end of the component is designated as the reference point and the sub-edit command is applied relative to this point. It is possible to delete, insert or replace characters within the component. For instance, consider the following component text:

All in a hott and copper sky.

To correct the spelling mistake type 12

```
proc start
  REM Save this listing as 'ArchEd3_prg'.
  w2a1ed
  run "ArchEd2"
  endproc
proc w1z2head
  print at 0,0; paper 2;" A R C H I V E  E D I T O R ";
  print rept(" ",mod*3):"Memory ";memory();" Bytes"
  if memory()<1200
    print at 1,mod*3;"**WARNING** Memory low - Close & backup file!"
    endif
  endproc
proc w2a1ed
  REM "ARCHIVE EDITOR - 'ARCHED' Version 2.02 3/5/91"
  REM Copyright S.G.Mitchell
  local ln,i2,m,n,ft,z,upd,go1
  let i2=0
  let n=0
  let ft=numfld()
  while n<ft
    let m=n
    if fieldt(n)
      let s$=fieldv(n)
      let upd=0
      let go1=1
      while go1
        cls
        ink 4
        w1z2head
        print at 19,0; ink 2;"ENTER"; ink 4;": Edit field (";
        print ink 2;"i/d/r/di"; ink 4;"=insert/delete/replace/del&ins);"
        print at 20,0; ink 2;"(SHIFT)1,2,..,"; ink 4;"1"; ink 2;"0";
        print ": Accept field edit & skip (-)n fields (cyclic);"
        print at 21,0; ink 2;"F4/F5";
        print ": Abandon/Commit all field edits to current record & Exit;"
        print at 22,0;"Rp "; ink 2;"SPACE ENTER";
        print ": Re-edit current field when at next component;"
        print at 23,0;"   "; ink 2;"SPACE SPACE ENTER";
        print ": Accept field edit  ""    ""    ""    ""  .";
        ink 6
        let line=1
        w2e1zm
        w2b1disp:n,s$,1
        w2f1zn
        if n+1<ft
          if fieldt(n+1)
            let ln=line
            let line=line+mod-6
            w2b1disp:n+1,fieldv(n+1),0
```

```
            let line=ln
            endif : endif
        print at 18,0;" "
        let z=0
        while not z
          let z=code(getkey())
          if z=30
            w2c1edit:2
            w2d1as
            if a$=fieldv(n)
              let upd=0
              else
              let upd=1
              endif
            if p3 or a$=" "
               let go1=0
               if n=ft-1: let m=-1: endif
               endif
             else
             if z=22
               let go1=0: let m=ft
               else
               if z=21
                 let go1=0: let upd-0: let m=ft
                 else
                 if 47<z and z<58
                   let go1=0
                   if z=48: let z=58: endif
                   let m=n+z-49
                   if m>=ft-1: let m=m-ft: endif
                   else
                   if z=33 or z=64 or z=35 or z=36 or z=37 or
                     z=94 or z=38 or z=42 or z=40 or z=41
                     let go1=0
                     if z=33: let m=n-2: endif
                     if z=64: let m=n-3: endif
                     if z=35: let m=n-4: endif
                     if z=36: let m=n-5: endif
                     if z=37: let m=n-6: endif
                     if z=94: let m=n-7: endif
                     if z=38: let m=n-8: endif
                     if z=42: let m=n-9: endif
                     if z=40: let m=n-10: endif
                     if z=41: let m=n-11: endif
                     if m<-1: let m=ft+m: endif
                     else
                     let z=0
                     endif : endif : endif : endif : endif
          endwhile
        endwhile
      if upd:w2g1af;n: let i2=1: endif
      endif
    let n=m+1
    endwhile
  if z=21 and i2: back : next : let i2=0: endif
  if i2: update : endif
  w2e1zm
  w2f1zn
  let a$=""
  endproc
proc w2b1disp;x,x$,a
  local fl1,k,m,n,q,l,lp,x1$
  let fl1=fl-1
  if a: let p4=1: let q=2
    else
    let q=1
    endif
  print at line,0; ink 4;fieldn(x)+":";
  let lp=len(x$)
  if lp=0: let x$=" ": let lp=1: endif
  let m=1
  while m<=lp
    let n=m
    if n+fl1>lp
      let l=lp-n
      let x1$=x$(n to n+l)
      print at line,col;x1$
      let line=line+2
      if a
        w2b2am;p4,x1$
        let p4=p4+1
        return
        endif
      let l=fl1
      if x$(n+1)<>" "
        while x$(n+l-1)<>" "
          let l=l-1
          endwhile
        let m=n+l
        let l=l-1
        else
        let m=n+l+1
        let k=lp
        while m<=k
          if x$(m)=" ": let m=m+1
            else : let k=m-1: endif
          endwhile
        endif
      while x$(n+l-1)=" "
```

spaces, denoted by dots, followed by a 'd':

All in a hott and copper sky
.....d^

and then press ENTER. The character above the 'd' will be deleted when the file is re-displayed; in this case the second 't' is deleted. Each of the sub-edit options is performed in much the same way. These are now described.

It is possible to delete a whole string of characters from a component by typing a set of consecutive 'd's. The words 'planks did swell' will be deleted from the following text:-

And all the planks did swell;
.....ddddddddddd^

To insert text in a component the cursor is again positioned relative to the reference point. An 'i' is placed underneath the character in front of which new text is to be inserted. For instance the words 'boards did shrink' will be inserted to the right of the semi-colon in the following text:

And all the ;
.....iboards did shrink^

Inserting text is the default for sub-edit commands. For example:

And all the;
.....boards did shrink^

would have the same effect. The only exceptions to this are if the first character to be inserted is a 'd', 'i' or an 'r'. In these cases the initial 'i' is then obligatory.

Text of equal length can be replaced character-for-character by placing the cursor underneath the first character to be replaced and typing an 'r'. The replacing text is then typed before pressing ENTER. In the following example the word 'below' is replaced by the word 'above':

Right up below the mast did stand.
.....rabove^

A more usual requirement is to replace text of unequal lengths. This can of course be performed as two separate operations, viz, first be deleting the old tex and then by inserting the new (or vice-versa). However, it is possible to achieve this in one operation. The text to be replaced is first deleted, as described above, but before pressing ENTER an 'i' is typed followed by the replacing text. For example, the words 'Not much larger' are replaced by the words 'No bigger' in the following text:

Not much larger than the moon.
ddddddddddddiNo bigger^

Sub-edits are performed for each component of the current field. If a particular component requires no edit then simply press ENTER. The cursor will move to the

start of the next component. Once the last component has been edited and ENTER pressed the current field will be displayed in its edited form. The field in this new form can be edited again by pressing ENTER and performing another set of sub-edits.

This process can be repeated any number of times until the full text of the current field is correct. The numeric key 1 is then pressed to move to the next textual field in the record. Larger jumps of from two to 10 fields can be made relative to the current filed by pressing the appropriate numeric key. To skip 10 fields use key 0. ArchEd considers the last field in the current record to be followed by the first field of the same record. As such the fields are arranged clockwise in a circle. Clockwise is synonymous with positive progress around the fields. Pressing a numeric key that would skip beyond the last field will give a new current field at an appropriate number of fields from the beginning of the record. For example:

Numbered fields on record = 12; current field = 10th; press 5; current field becomes 3rd. (If the third field is numeric the next textual field becomes the current field).

Similarly, pressing a SHIFTed/Key will move ArchEd anti-clockwise, with the last field following the first. This is the negative direction, and a SHIFTed numeric is taken to mean minus that number of fields from the current field. Whatever numeric key is pressed, the edits to the current field are accepted and will be applied if the record is updated – see below.

Movement

Movement around the record in either direction can be repeated. During this cycling individual field edits can be 'accepted' any number of times. Once the record is considered to be correct it is ready to be updated.

There are two 'rapid progress' options available which allows you to jump out of sub-editing after any particular sub-edit, to any particular component. Both these facilities are useful once a certain amount of practice and familiarity with ArchEd has been gained. The first option causes immediate re-display of the current field with any sub-edits applied while the second accepts any sub-edits and moves immediately to the next field. A single space is used in the first case and two spaces in the second. To do this, a sub-edit to a component is first completed by pressing ENTER. The cursor will move to the reference point of the next component as normal. If one space (or two, for the second option) is now typed and ENTER pressed, the appropriate action will occur. These options are obviously not applicable if the current field comprises only a single component.

```
        let l=l-1
        endwhile
      let x1$=x$(n to n+1)
      print at line,col;x1$
      let line=line+q
      if a
        w2b2am;p4,x1$
        let p4=p4+1
        endif
      endwhile
  endproc
proc w2b2am;n,x1$
  if n=1: let m1$=x1$
    else : if n=2: let m2$=x1$
      else : if n=3: let m3$=x1$
        else : if n=4: let m4$=x1$
          else : if n=5: let m5$=x1$
            else : if n=6: let m6$=x1$
              endif : endif : endif
        endif : endif : endif
  endproc
proc w2c1edit;row
  local l,m,n,go,e$,v$,w$,r1$,r2$
  let p3=0
  let n=1
  while n<p4
    if n=1: let r1$=m1$
      else : if n=2: let r1$=m2$
        else : if n=3: let r1$=m3$
          else : if n=4: let r1$=m4$
            else : if n=5: let r1$=m5$
              else : if n=6: let r1$=m6$
                endif : endif : endif
        endif : endif : endif
    let m=row+2*n-2
    let go=1
    while go
      input at m,col;e$
      let l=len(e$)
      if l
        if e$<>" " and e$<>"  "
          w2c2type;e$,"  d"
          if p1: let go=0
            if p1>1: let v$=r1$(1 to p1-1)
              else : let v$="": endif
            let e$=e$(p1 to )
            let p2=len(e$)
          let l=instr\e$,"i")
          if l
            let p2=l-1
            let e$=e$(l+1 to )
            else : let e$="": endif
          if len(r1$)<p1+p2: let w$=""
            else : let w$=r1$(p1+p2 to ): endif
          if l
            let p1=255-len(r1$)-p2
            if p1<256
              let e$=e$(1 to p1)
              endif : endif
          let r2$=v$+e$+w$
          if r2$="": let r2$=" ": endif
          w2c3an;n,r2$
          else
          w2c2type;e$,"  r"
          if p1: let go=0
            if p1>1: let v$=r1$(1 to p1-1)
              else : let v$="": endif
            let e$=e$(p1+1 to )
            let p2=len(e$)
            if len(r1$)<p1+p2: let w$=""
              else : let w$=r1$(p1+p2 to ): endif
            let r2$=v$+e$+w$
            w2c3an;n,r2$
            else
            w2c2type;e$,"  i"
            if not p1
              let l=1: let p2=len(e$)
              if p2=255: let p2=254: endif
              while l<=p2
                if e$(l)=" ": let l=l+1
                  else : let p1=l: let e$="i"+e$(1 to p2): let l=p2+1: endif
                endwhile
              endif
            if p1: let go=0
              if p1>1: let v$=r1$(1 to p1-1)
                else : let v$="": endif
              let e$=e$(p1+1 to )
              if len(r1$)<p1: let w$=""
                else : let w$=r1$(p1 to ): endif
              let p1=255-len(r1$)
              let e$=e$(1 to p1)
              let r2$=v$+e$+w$
              w2c3an;n,r2$
              endif : endif : endif
        else
        let go=0: let n=p4
        if e$="  ": let p3=1: endif
        endif
      else
```

```
            endif
          endwhile
        let n=n+1
        endwhile
      endproc
    proc w2c2type;x1$,x2$
      let p1=instr(x1$,x2$)
      if p1: let p1=p1+2: return : endif
      if x1$(1)=x2$(3): let p1=1: return : endif
      if x1$(1 to 2)=x2$(2 to 3): let p1=2: return : endif
      let p1=0
      endproc
    proc w2c3an;n,x1$
    if n=1: let n1$=x1$
      else : if n=2: let n2$=x1$
        else : if n=3: let n3$=x1$
          else : if n=4: let n4$=x1$
            else : if n=5: let n5$=x1$
              else : if n=6: let n6$=x1$
                endif : endif : endif
          endif : endif : endif
      endproc
    proc w2d1as
      local n,k
      if len(n1$): let m1$=n1$: endif
      if len(n2$): let m2$=n2$: endif
      if len(n3$): let m3$=n3$: endif
      if len(n4$): let m4$=n4$: endif
      if len(n5$): let m5$=n5$: endif
      if len(n6$): let m6$=n6$: endif
      let k=255
      let p2=len(m1$)
      if p2<k
        let p1=p2+len(m2$)
        if p1<k
          let p2=p1
          let p1=p2+len(m3$)
          if p1<k
            let p2=p1
            let p1=len(m4$)
            if p1
              let p1=p2+p1
              if p1<k
                let p2=p1
                let p1=p2+len(m5$)
                if p1<k
                  let p2=p1
                  let p1=p2+len(m6$)
                  if p1<k
                    let s$=m1$+m2$+m3$+m4$+m5$+m6$
                    else : let s$=m1$+m2$+m3$+m4$+m5$+m6$(1 to k-p2)
                    endif
                  else : let s$=m1$+m2$+m3$+m4$+m5$(1 to k-p2)
                  endif
                else : let s$=m1$+m2$+m3$+m4$(1 to k-p2)
                endif
              else : let s$=m1$+m2$+m3$
              endif
            else : let s$=m1$+m2$(1 to k-p2)
            endif
          else : let s$=m1$+m2$(1 to k-p2)
          endif
        else : let s$=m1$
        endif
      endproc
    proc w2e1zm
      let m1$="": let m2$="": let m3$=""
      let m4$="": let m5$="": let m6$=""
      endproc
    proc w2f1zn
      let n1$="": let n2$="": let n3$=""
      let n4$="": let n5$="": let n6$=""
      endproc
    proc w2g1af:n
      if n<3
        if n=1: let field1$=s$
          else : let field2$=s$: endif
        else
        if n=3: let field3$=s$
        else : let field4$=s$: endif
      endif
    endproc
  proc w2g2af:n
    if n<4
      if n=0: let Title$=s$: endif
      if n=1: let Fname$=s$: endif
      if n=2: let Sname$=s$: endif
      if n=3: let Surname$=s$: endif
      else : if n<9
        rem n=4 'Num', omitted numeric field.
        if n=5: let House$=s$: endif
        if n=6: let Street$=s$: endif
        if n=7: let Town$=s$: endif
        if n=8: let County$=s$: endif
        else
        if n=9: let Post$=s$: endif
        if n=10: let Country$=s$: endif
        rem n=11 'Index', another numeric field.
        if n=12: let Phone$=s$: endif
        if n=13: let Key$=s$: endif
        endif : endif
    endproc
```

A 'quit' function is provided by pressing key F4 which abandons all the edits performed for all the fields of the current records during this editing phase. This would normally only be chosen if it was suddenly realised that the wrong record was being edited. A record is updated (ie the edits committed) if key F5 is pressed between field edits. A quit or an update causes a return to be made to the record selection process.

A few words of caution are now given, both on the use of ArchEd and the subsequent use put to an 'ArchEd' edited file.

1. There are two instances in procedure w2b1disp where an unrecoverable 'string subscript' error (Error 77) can occur, causing ArchEd to fail. Both these errors arise when the procedure is performing a backward search along the current field to locate a particular character – the first instance involving a search for a space and the second for a non-space. *The error is avoided if fields never contain strings of 48 (or more) consecutive spaces or non-spaces.*

In standard text this is never the case, so this error should never be a problem. If the error occurs processing control immediately passes from the procedure to the standard command level of Archive. All but the last field edit to the current record can be applied if an Archive update command is given immediately. The update must be followed by a 'close' on the file before ArchEd or your own procedures are restarted.

2. It is possible to insert a string of (up to) the full length of the Archive input buffer into the current component. The same is true when replacing text although once the component length has been exceeded the excess replacing text is treated as insertion text. In both these cases any following components, starting with the last, will be pushed off the end of the field to accommodate the inserted text. However, inserting such long strings tends to be impractical since the inserting text tends to obliterate the text of the following components as shown on the tv monitor screen. It is better to insert part of the text, redisplay, and then insert the rest of the text.

3. The Archive 'alter' command restricts field displays to less than screen width in much the same way that sedit fields were originally restricted to screen width. Excess text over-writes the initial text of an 'alter' field and it is almost impossible to assess how CTRL&<- or CTRL&-> changes affect the field data. It is there fore better not to use 'alter' on ArchEd edited files but to use only your own application procedures or ArchEd to make further changes.

The following descriptions can be skipped if you're not interested in the inner workings of the 'ArchEd' Procedures.

The w2 editing procedures have the following structure and function:

1. a1ed is the controlling procedure for editing an individual record from the file. Two 'while' loops control the presentation

of each field of that record, via a work variables s$, to two subordinate procedures. The first of these procedures is b1disp, whose task is to split the current field into components and display these on the screen. The second, c1edit, is the heart of ArchEd, and handles the editing of the component strings of the current field. Upon return from a c1edit a call to d1 as sets the value of s$ to the 'after' images (see 3 and 4 below) resulting from the edit. Upon exit from the inner 'while' loop, and given that a meaningful edit has been applied, s$ is assigned to the current field by a call to g1af. Dependent upon the type of exit from the outer 'while' loop the current record is updated. Finally, the component strings and s$ (all global variables) are set to 'null' in order to save space upon exit from Arch Ed.

2. b1disp has two main tasks. These are firstly to split up the current field into component strings and secondly to display them, double-line spaced on the screen. The procedure is also used to display the field following the current field, this time single-line spaced.

3. b2am is called by b1disp to assign the component strings to the six string variables m1$-m6$. These variables are the 'before' images of the field. Six other variables, n1$-n6$, are used during the editing process as the 'after' images of the field.

4. c1edit controls the editing of an individual field from the current record. Two 'while' loops handle the editing of the component strings. The first selects the next component and assigns it to a work string (r$). The second both validates and actions the sub-edit commands (i/d/r - insert/delete/replace) to the work string. A call to c2type assists in this task. The edited work string is assigned to the appropriate 'after' image variable by a call to c3an.

Convoluted

There were a number of problems I encountered while writing these procedures. These mostly stem from the somewhat restrictive nature of Archive's particular dialect of Basic.

Not having array processing at my disposal has lead to some necessarily convoluted techniques to achieve the desired results. These techniques have themselves contributed to the slowness of the procedures in their efforts to perform the edits. Certainly if improvements to Archive Basic were forthcoming from Psion these procedures would undoubtedly benefit both in an increase in speed of operation and a cut-back in the amount of storage they occupy. For instance, the ability to assign a value/string directly back to a particular (non-named) occurrence of a field as in, for example:

'let fieldv(n)=x$'.

However, this is not a valid assignment statement in Archive. Examination of the procedures will show the protracted nature of the coding Archive labours through to achieve this simple task.

You'll also notice that I've made no use of the error command. During development I noticed that when such error-governed failures occurred memory used by the error trapped procedures was not fully released for re-use. Designing the procedures based on the error command, to control certain processing, led to a gradual loss of memory – a sort of 'creeping amnesia', one might say. I have there fore outlawed their use in procedures what I've written other than where memory was never likely to be a problem and then only in cases of true fatal error processing.

Having stated these criticisms I do however recognise the restrictions in developing a tool like Archive to suit all requirements and accept that the many facilities within Archive represent terrific value especially when one considers that Archive was bundled in with the QL. Any improvements that were to have come Archive's way given that it was to be continually developed, would no doubt have negated the need for ArchEd; a slicker way of handling text strings would be now (no doubt) have existed.

Hopefully, if you have had any difficulties in using Archive, especially in this area of text handing, then you will turn back to it afresh – armed with ArchEd at your fingertips.

NEW FROM DIGITAL PRECISION

CPORT BASIC TO C PROGRAM TRANSLATION SYSTEM
This program translates SuperBASIC programs directly into C source code, automatically! If you want to move programs into C for migration to other hardware, or want to get your programs running faster, or simply want to learn C the easy way (chuck BASIC in one end and examine the C that spews out the other), CPORT is the system for you. CPORT is extremely friendly, easy-to-use and tolerant of poorly-written BASIC. There is even a method of dealing with BASIC toolkits. The C it will generate is very readable, human-like and is often optimal. Of course, the better the quality of the BASIC you put in, the better the clarity of the C that will be generated. But don't misunderstand - even if your BASIC is a rats-nest of GOTOs and GOSUBSs, CPORT won't mind. Usually, the generated C - which can even be switched between the ANSI and Lattice (K&Rish) industry standards - needs no tinkering with. The only conditions worth mentioning are that there must be no computed branching (e.g. GOTO 3*Y+L), no interpreter-only commands (LIST, EDIT, RENUM etc) and that if the program contains any PROCs or FNs (as it probably will), there mustn't be a GOSUB *as well* - not restrictions at all. CPORT is an amazing program, making breakthroughs in AI. CPORT is available on its own or together with the acclaimed C68 compiler.

SUPERBASIC MONITOR
Yes - this program monitors and reports on the performance of SuperBASIC programs as they run (i.e. dynamically) under the interpreter. Even if you only occasionally tinker, this one you must have! Ideal for use with XREF, BETTER BASIC, TURBO etc.

XREF v2.0
An incredibly competent program analyser - structure, the dynamic call hierarchy of procedures/functions, step-ladder report, glossary, warnings, variable usage and so on. Ideal with SUPERBASIC MONITOR, BETTER BASIC, TURBO etc.

QMON v2.05
The ultimate version of Tony Tebby's superb machine-code Monitor. An absolute must for those who really want to know what's going on in the QL. £10 off if you return the old Digital Precision Monitor.

COMPARE
This little gem compares files (data or program) at great speed, and allows shuffles and alignment in auto- and semi-auto mode. You cannot do without it!

MEGA DICTIONARY
If you have over 1.5Mb RAM (Goldcard, some ST/Thor) this is the ultimate PERFECTION PLUS accessory, enabling the best possible spellchecking. It contains approximately 360,000 words, which really says it all. Another attraction of this masterpiece - we will soon be announcing a suite of programs enabling sophisticated user-controlled access to the dictionaries and giving previously unsurpassed power at all sorts of word-handling (crosswords, anagrams, missing letters/groups, properties, all sorts of board and TV games). So get the Mega Dictionary now!

SPELLCHECKER
SPELLCHECKER works with and without PERFECTION. SPELLCHECKER can always spellcheck files - either PERFECTION format or plain ASCII (Quill or text87 export files, for example). If PERFECTION is present, SPELLCHECKER can also selectively (pages/blocks/all) spellcheck the current document, or spell as you type. Peak spellchecking speed is over 35,000 characters per second on Goldcard even with the Mega Dictionary, and a cracking 3 pages per second on ordinary QLs. SPELLCHECKER comes with two ready-made dictionaries (the bigger is for 640K or bigger setups) and a system for creating and maintaining (add/delete/edit/view) user dictionaries. Further, you can spellcheck using 1 or 2 dictionaries - typically a supplied one and a user one - you can even specify which is to be used first when checking! So if you bought PERFECTION without SPELLCHECKER and now want to add it, or even if you do not have PERFECTION at all, this is the product for you!

QMATHS MATHEMATICAL SYSTEM
An incredible mathematical compendium for the QL. Pride of place goes to the symbolic problem solver.... It can solve problems, simplify expressions, factorise, expand etc etc - all symbolically! If you could sneak this one into a maths examination (school/GCSE/O/A/S/undergraduate) you would have a formidable ally. It knows about all the algebraic operators, powers, roots, brackets (any number), trigonometry, matrices, determinants, vectors, factorials, perms and combs, binomials, exponentials, logarithms, hyperbolics, inverse functions, infinite series and their approximations, complex and imaginary numbers, conversions, and even calculus - both differential and integral (even knows definite integration, integration by parts etc)! And when the program is working something out, you can opt to get it to display some or all of the steps either All this is accompanied by a superb interactive tutorial. So whether you have been terrified of maths or are a boffin, this is the program for you: no mathematical skills are assumed or needed. Whether all you want to do is compute 2+2 or d/dx((sin(x)+x.log(x))^(x^(g(x)))), QMATHS will do it. There is nothing like this available on any computer. In addition to this program, the package also contains an interpretive, fractal image-generating language with loads of beautiful fractal programs supplied for you to use, modify or adapt. No programming skill is assumed or needed. In addition, there is a multiple precision floating point maths package - allowing calculations with all the QL functions at precisions up to over 600 decimal digit accuracy (that is not a misprint) and very fast too. In addition, there is a 3D surface modelling program and lots of calculating routines to perform all sorts of algebraic and statistical computations in your own BASIC or Abacus systems. This is an incredible package.

PROFESSIONAL PUBLISHER TOOLBOX PART TWO
Did you think we'd stopped? Another blisteringly good collection of fonts and utilities for the Professional Publisher user, augmenting and adding to the first toolbox. You really should have both.

HARDBACK + FINDER
THE hard disk utility.

RECOVER
Recovers lost Archive databases. Ideal companion for Media Manager Special Edition.

ARCHDEV + RTM
The Archive development environment - gives enhanced speed, greater workspace and a cleaner boot system.

DATABASE ANALYSER
Fast statistics about your Archive database(s).

ARCHIVE TUTORIAL
Everything you always wanted to know about Archive - but were afraid to ask.

NAMES + ADDRESSES
MAILMERGE
DAT-APPOINT
Names and addresses, mailmerging and appointment diaries - sophisticated, fast and ready to run under Archive. If you've never used your Archive, now is the time to start!

SEDIT
SCREENPRINT
Creates/edits and prints screen-format files in Archive. An invaluable aid.

PEDIT
A fast, sophisticated printer driver manager for the Psion programs - replaces the free one!

CASH TRADER v3.0 INCLUDING ANALYSER
PAYROLL
An accounts system for the small and medium-sized business, with lots of excellent reporting and management facilities. Aimed at the layman.

NEW!

PLEASE SEE PAGE 10 FOR SPECIAL DEALS (YOU CAN SAVE UP TO 25% ON THESE AND OTHER DP PROGRAMS), OTHER DP PRICES AND OUR TERMS & CONDITIONS.

CPORT WITH C68 COMPILER	99.95	cT
CPORT BASIC TO C TRANSLATION SYSTEM	89.95	cT
SUPERBASIC MONITOR	24.95	aT
XREF v2.0	29.95	aT
QMON MACHINE CODE MONITOR v2.05	39.95	at
COMPARE	19.95	aT
MEGA DICTIONARY	29.95	gT
SPELLCHECKER	49.95	dT
QMATHS MATHEMATICAL SYSTEM	59.95	cT
PROFESSIONAL PUBLISHER TOOLBOX PART TWO	29.95	eT
PROFESSIONAL PUBLISHER TOOLBOXES (PAIR)	49.95	eT
HARDBACK + FINDER	49.95	cT
RECOVER	19.95	aT
ARCHDEV + RTM	29.95	aT
ARCHIVE TUTORIAL	19.95	aT
DATABASE ANALYSER	19.95	aT
NAMES + ADDRESSES	19.95	dT
MAILMERGE	19.95	aT
DAT-APPOINT	19.95	dT
SEDIT	29.95	aT
SCREENPRINT	19.95	aT
PEDIT	19.95	aT
CASH TRADER v3.0 + ANALYSER	99.95	cT
PAYROLL	49.95	cT
! Minimum 1.5Mb RAM:only available on disk		g

Note: Some of the above had been available from PDQL previously (usually earlier versions)
Once-recorded cartridges £50/25; £75/50; £100/100
Disk interfaces (6 months guarantee) only £39.95!